Abstract

Title of dissertation	Interactive Event Sequence Query and Transformation
	Megan Monroe, Doctor of Philosophy, 2014
Directed by	Professor Ben Shneiderman Department of Computer Science

In our burgeoning world of pervasive sensors and affordable data storage, records of timestamped events are being produced across nearly every domain of personal and professional computing. This *temporal event data* is a fundamental component of electronic health records, process logs, sports analytics, and more. Across all domains, however, are two overarching needs: (1) to understand population-level trends and patterns, and (2) to identify important subsets of individual records.

Visual analytics tools are billed as the solution to both of these problems. A huge volume of work has demonstrated the ability of these tools to facilitate user-guided data exploration and hypothesis generation across a wide range of data types. What is typically ignored however, is the process that takes place between the data collection and this exploration stage, a process frequently referred to as data wrangling. For many data types, wrangling consists mostly of restructuring spreadsheet columns and renaming fields. For temporal event data though, this wrangling process can extend much further—to the data itself—where event patterns must be transformed to better reflect either the real world events that generated them or the perspective of a given study. Without this step, population-level trends can be obscured beyond the point of recognition, and important subsets of records are impossible to discern.

Temporal event data wrangling, however, is deceivingly difficult and error prone even for expert users. Standard, command-based query languages are poorly suited

for specifying even the simplest event patterns and, in systems that are not precisely designed for handling temporal constructs, these queries are executed using a series of slow and inefficient self-join operations. Attempts at more accessible query languages frequently omit critical features such as events that occur over a period of time (intervals) or the absence of an event. Perhaps most importantly is that query alone is not enough to get users through a typical temporal event data wrangling process. Event patterns not only need to be found, but also transformed and re-represented. Temporal event wrangling is just as much about revisal as it is about retrieval, and given the ubiquity of this data type, an effective solution on this front has the potential to hugely impact the way that we utilize this data to inform future decisions. An improved query and wrangling process would not only benefit database professionals, but also dramatically increase the range of users who can access this type of data, particularly domain expert medical researchers.

This dissertation demonstrates the ability of the EventFlow visualization tool to extend beyond the typical bounds of data exploration, and serve as a critical aid for both temporal event query and data transformation. I begin by establishing a better understanding of why these two processes are innately error prone, and introduce a simple set of powerful yet usable mechanisms that can help reduce an initial portion of these errors. I then show that by coupling these mechanisms with interactive visualizations, users are able to both identify remaining errors and leverage those errors to construct more accurate queries and transformations. The direct contributions of this dissertation are (1) a graphic-based query capabilities over points, intervals, and absences, (2) an integer programming strategy for processing temporal queries, (3) a Find & Replace system for transforming event sequences, and (4) eight case studies that demonstrate the utility and validity of these approaches. However, this work is designed more broadly to open new avenues of research in how visualization and visual analytics tools can be leveraged for tasks beyond data exploration.

Interactive Event Sequence Query and Transformation

by

Megan Monroe

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2014

Advisory Committee:

Professor Ben Shneiderman, Chair
Dr. Catherine Plaisant
Professor Bill Gasarch
Professor Amol Deshpande
Professor Jon Froehlich
Professor Jen Golbeck

To Club Cliff,
For the clear skies, and calm waters.

Acknowledgments

Out of a 300+ page dissertation, these are the two pages where I can write whatever I'd like, so let's make them count, yes? I would like to begin by thanking my advisors, Ben Shneiderman and Catherine Plaisant. How is it that you came to find me at the exact moment that I began searching for a research project? How were two, perfectly sane individuals such as yourselves able to teach and support and collaborate with a bizarre creature like me? How did it all seem so enjoyable, so effortless?

(a) I cheated.

(b) I'm lucky.

(c) I'm a genius.

(d) It is written.

Thank you, Ben and Catherine, for your time and energy and trust. Thank you for arguing with me and ordering Chinese food and making sure that I consistently used the plural form of "user." You have made me harder, better, faster, and stronger, and no less daft to boot. Ben, thank you for always remembering everyone's name. Catherine, thank you for always being right.

I would like to thank my committee members, Bill Gasarch, Amol Deshpande, Jen Golbeck, and Jon Froehlich. As anyone who binge-watched *Orange is the New Black* in 24 hours would know (not me, of course—I was working), the right supporting cast can easily outshine the primary ensemble. You were all amazing contributors, collaborators, and characters. You helped to expand my research and my mind without ever leaving me holding the shrimp.

Now things get tricky. How do I even begin to compile and quantify the countless contributions and inspirations that have come from the HCIL, the Computer Science Department, the University of Maryland, and the DC community at large?

Only a grand gesture would suffice of course, but in the interest of time and space, a simple list will have to do. Thank you to Ben Bederson, Tanya Clement, Allison Druin, Anne Rose, Charley Lewittes, Jenny Story, Fatima Bangura, Dave Mount, Juan Morales del Olmo, Matt Mauriello, Raul David Guerra, Ramakrishna Padmanabhan, Krist Wongsuphasawat, Rongjian Lan, Allen Fong, Hanseung Lee, Fan Du, Chris Imbriano, Alex Quinn, Sana Malik, Jeff Milstein, Sigfried Gold, Seth Powsner, Gigi Lipori, Tamra Meyer, Rose Thelus, Beth Carter, John Gilmore, Xinggang Liu, Candice Young, and Sarah Parker. And a special thanks to John Alexis Guerra Gómez, Cody Dunn, and Sureyya Tarkan for reminding me that all of this has happened before, and all of this will happen again.

The final acknowledgements will go to my friends and family. First, to my parents: Bev, Kev, Kent, Pat, Bill, Cheryl, Jeff, Nancy, and Bob. That's right, I have nine parents. If you add it all up (make sure to carry the one), that's a whole lot of visits, phone calls, and care packages. These seemingly minor gestures add up and, in the end, this experience became more *Oliver & Company* than *Oliver Twist*. More singing. Less starving. The blue ribbon goes to Bev for the most inspiring (Whopper Eggs!) and inscrutable (Hope Solo Autobiography?) gifts of all, and for constantly telling me that I'm pretty, and funny, and smart - just like that chubby baby in *The Help*. And to all of you, thank you for giving so much, and asking only for Palo Alto Pepper Sauce in return.

Then we have the specials: Kady Lady, Suki, Spencer, Meanie, Mols, and Thommo. No one would actively choose to be friends with a Computer Science Ph.D. student, and none of you asked for this, but you tolerated the inconvenience gracefully. For the crêpes, and quesodillas, and Remedies. For fashion week, and date night, and the stoop. For attic DPs, and autopilot controls, and 2nd Wednesdays. For "bait and switch," and "acting out," and "basically fly themselves." For *The Hunger*, and *I Think We're Alone Now*, and *Overboard*. For the Toys, and Birds, and Angels. Life is an inside joke, and we're all in this together - probably holding hands.

And lastly, to Netflix. We understand each other. I think we could be together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Across all domains of research, there is an overarching need to understand the past. We intently observe that which has happened, hoping to outrun the doomed repetition of a history not understood. Fortunately, for anyone in need of a Cliffs Notes version, the vast majority of historical occurrences can be boiled down to a series of categorized, timestamped events. These time-event pairings, (e.g. Patient 0: 7/18/1983 - Birth), comprise datasets across a wide range of domains including Electronic Heath Records (EHRs), law enforcement response logs, online resource use, and sports reporting. The prevalence of this data type can be attributed to the ease with which it can be stored, the increasing ability to collect it automatically, and the simplicity of the three questions that it strives to answer:

- What happened?
- When?
- To whom?

Understanding this temporal event data requires comprehension on two different levels. The first of these, which is referred to as *intra-record understanding*, relates to the events that happen to a single entity. One might ask, for example, how long a patient had been taking a certain medication before he experienced a symptom. The second level of comprehension is referred to as *inter-record understanding*. Here, the focus is on population level trends across multiple entities. For example, across all of the patients who experienced the symptom, one might want to know how many of them had been taking that medication.

The difficulty of temporal event data, however, is that these seemingly simple questions can be obscured by the structure of the data. Data of any kind can often necessitate a potentially lengthy series of formatting changes - perhaps columns need to be sorted, or fields renamed. However, despite the time that these formatting issues can consume, they are relatively straightforward operations that can typically be accomplished using some clever trickery in Excel or some basic SQL code. On the other hand, temporal event data can contain complications that extend all the way to the events themselves, requiring that the actual, underlying data be altered in order to reveal critical features. This problem manifests itself in three common ways:

1. **The data does not match the real world events.** Consider the above example where a patient takes a medication for a given period of time, and then experiences a symptom. In the raw data, this medication will likely appear as a series of individual prescriptions. These individual prescriptions may be disjoint or overlapping, depending on when the patient happened to pick up each prescription. In reality though, unless there is a significant gap between prescriptions, the patient was exposed to the effects of the medication for a single, continuous interval of time. This duration of exposure is what will dictate subsequent symptoms, more so than the sequence of individual prescriptions.

2. **The data does not match the perspective of the study being conducted.** Consider a researcher who is trying to understand the number of doctor's visits that are necessitated by different types of ailments and injuries. The raw event data, however, might only include information about the type of doctor that the patient went to see on any one visit. This researcher would have to infer then, when a patient visits a hand specialist the day after visiting a general practitioner, that the patient has had two doctor's visits regarding a hand injury.

3. **The data is unnecessarily cluttered.** Consider a dataset consisting of thousands of patients who experienced a given symptom multiple times. The most critical feature of each of these patient records, however, might be the time at which the symptom first presented. In this case, every other symptom event in the dataset is extraneous. Only the first symptom in each record is needed to proceed with analysis.

Changes like these, to the underlying event data, are not the designed functionality of current visualization or standard statistical querying tools. Visualization tools frequently operate from an ivory tower: allowing users to better explore their datasets, but under the assumption that the data has arrived in an explorable condition. When this is not the case, these tools are more or less useless, producing unintelligible displays, such as the example in Figure 1.1, that are impossible to gain any meaningful insight from. Users are rarely provided with any tools for cleaning the data beyond basic filtering options.

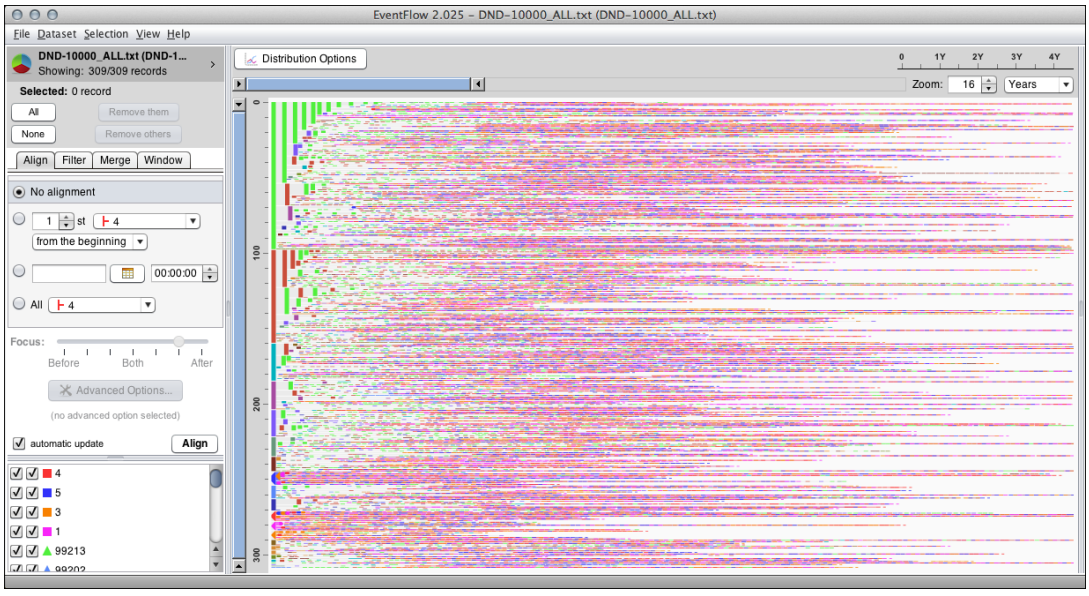


Figure 1.1: The EventFlow visualization tool, loaded with a raw temporal event dataset, can create displays that are too complex to effectively explore.

By contrast, statistical querying tools can be more effective at powering through complex event patterns, but with three significant drawbacks. The first is that these command-based query languages are prohibitively difficult to learn. It is not reasonable to expect a wide range of users to be able to specify a query such as the example in Figure 1.2. Second, even for expert users, it is extremely easy to make errors in the specification of such a query and, with no visual feedback about the query execution, for those errors to go unnoticed. Finally, command-based querying relies on the assumption that users know exactly what they are looking for. Event sequences that are unexpected or unusual frequently go unnoticed.

```
select distinct t1.patid,
       t1.drug,
       t1.dispense_date,
       t1.next_drug,
       t1.next_dispense_date
from (select distinct patid,
       dispense_date,
       lead(dispense_date,1) over (order by patid, dispense_date, drug
          ) next_dispense_date,
       drug,
       lead(drug,1) over (order by patid, dispense_date, drug)
          next_drug
from  DRUG_TBL
where drug in('DRUG A', 'DRUG B')) t1,
       EVENT t2
where t1.patid = t2.patid and
       t2.ICD9 = 'STROKE'      and
  ((t1.drug = 'DRUG A' and t1.next_drug = 'DRUG B') and
          (t1.dispense_date = t1.next_dispense_date) and
          (t1.next_dispense_date < t2.event_start or  t1.
             dispense_date > t2.event_end));
```

Figure 1.2: The SQL specification for a relatively straightforward sequence of overlapping events.

With failings in both domains of analytical tools, there remains a pressing need for users to be able to make informed changes to large datasets of temporal event records. This dissertation addresses that need by extending the EventFlow visualization tool to also function as a data transformation application. EventFlow,

shown in Figure 1.3, was the ideal candidate for such an extension due to its focus on both intra-record and inter-record understanding. The crux of the application is a set of dual, interactive displays: one that depicts each individual event record in a scrollable list, and one that depicts an aggregated view of the entire dataset in a single window (see Appendix A). The latter provides users with a holistic view of how complex operations such as queries and transformations affect the dataset, and the former allows users to investigate the detailed nuances that can lead to both the intended and unintended results of these queries and transformations.

Overall, I demonstrate that, by leveraging visualization throughout the data cleaning process, users can generate datasets that more accurately represent their intended research objectives. These transformed datasets not only allow for the standard, visual analytics process to proceed without impediment, but also serve as an effective means of communication between collaborators across various domains. Visualization should help users to tell a story. It should serve as the communicative link between our natural way of thinking and reasoning, and the complex, computational underpinnings of big data. The goal of this work is to show that this is less of a natural fluency and more of an intricate translation that requires user-driven knowledge, well-designed interaction tools, and interpretable feedback.

1.1 Contributions

The direct contributions of this dissertation are as follows:

Graphic-based query capabilities over points, intervals, and absences:

Command-based query systems offer access to a wide range of temporal relationships, but have a prohibitively steep learning curve. By contrast, graphic-based query systems are easier to learn, but are frequently limited in scope. Critical temporal phenomena, such as intervals and absences, are often omitted. These

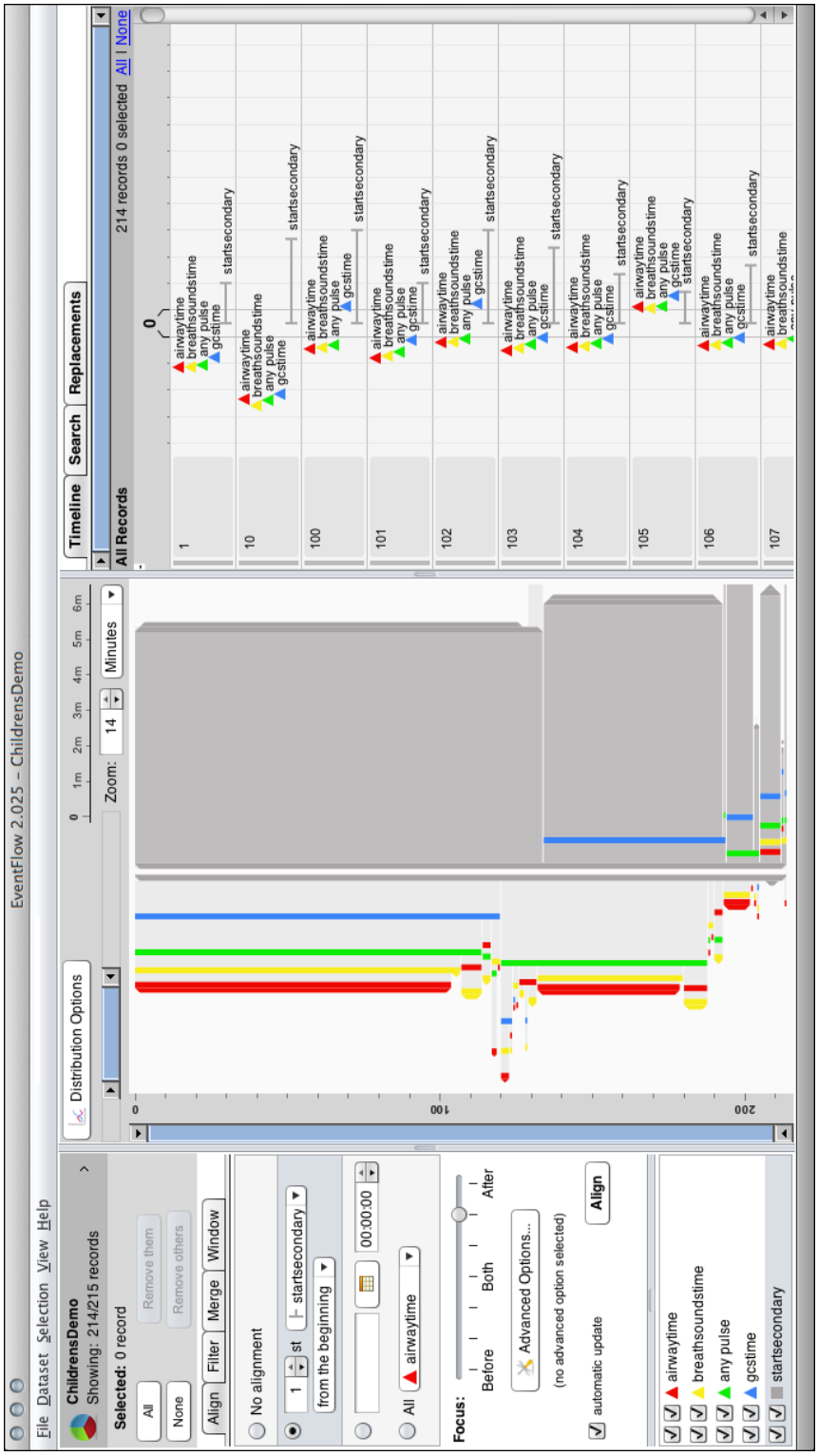


Figure 1.3: EventFlow consists of three panels: the control panel (left), the aggregated record display, which depicts the entire dataset in a single window (center), and the individual record display, which depicts each individual event record in a scrollable list (right).

complex phenomena, however, are the most difficult to specify accurately, and users could benefit substantially from more accessible, graphical representations and visual feedback. To this end, I make the following contributions:

- The identification of common difficulties that users encounter when specifying queries involving complex temporal phenomena such as intervals and absences.
- A set of easy to learn graphical representations for a wide range of temporal events and event relationships.
- A basic query interface that allows users to easily specify common event relationships.
- An advanced query interface that captures any event sequence that can be expressed as a finite sequence of inequalities, including the comprehensive list of Allen's interval event relationships, enhancements that account for point-based events, and both point and interval absences (see Chapter 4, Figure 4.22). This scope is then further extended to support metric temporal constraints, event attribute constraints, and common shortcuts taken from string matching tactics.
- An integration of query and visualization that better accounts for every stage of the overall search process (Figure 1.4).

An integer programming strategy for processing temporal queries:

Standard, relational query processing is notoriously ill-suited to temporal event queries. Without extensive data preprocessing, or a database system with custom, temporal constructs, these queries result in a prohibitive number of large, self-join operations. Sequential query processing strategies, such as those used in string matching, also become increasingly complex as you move away from exact match

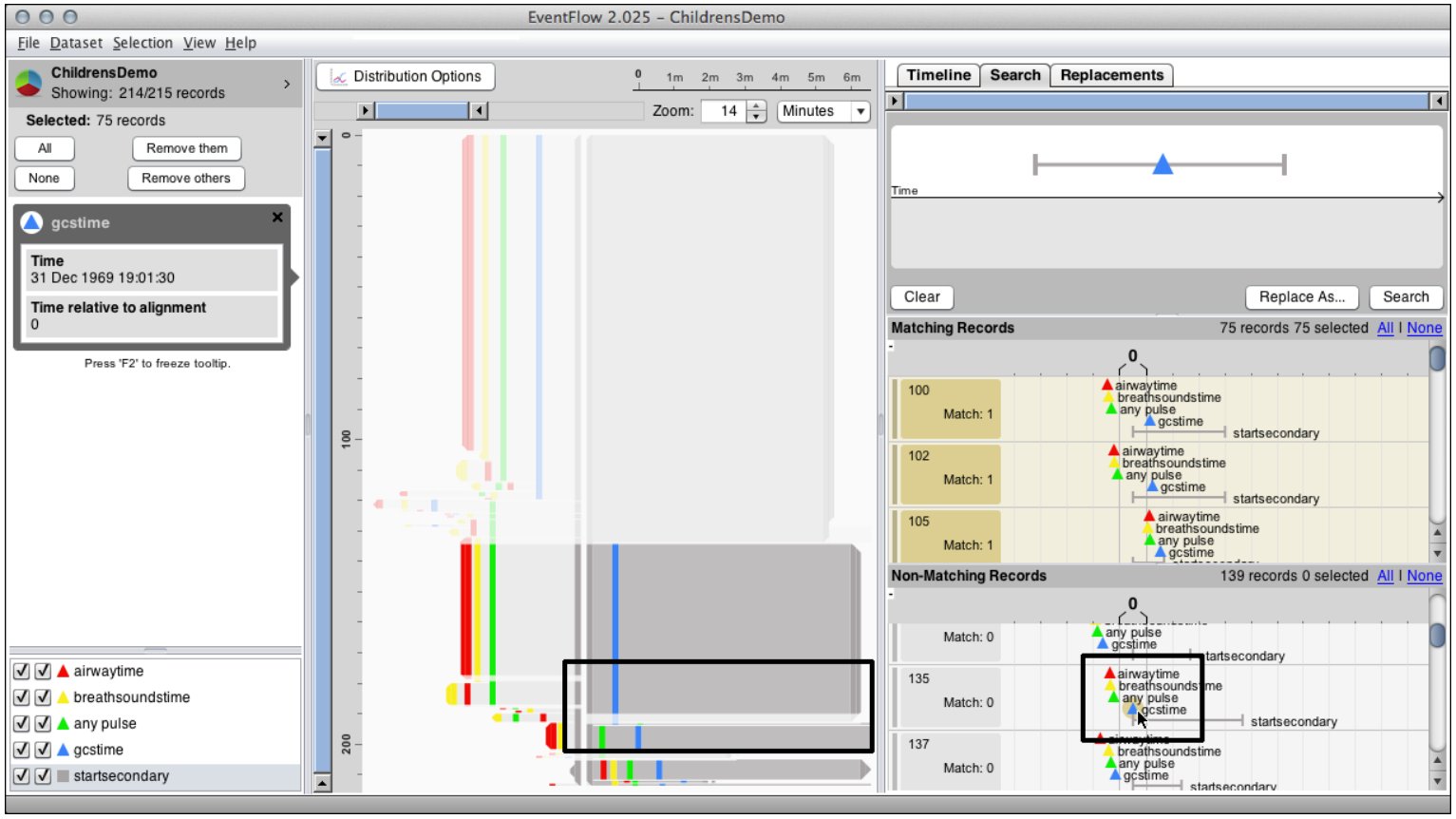


Figure 1.4: Shown here is EventFlow's advanced query interface in the rightmost panel including the query specification canvas (top), the matching results (middle), and the non-matching results (bottom). By integrating visualization and query, possible false negatives are visible in the aggregated record display. Using the more detailed, individual record display, it can determined that, in some cases, the blue event exactly coincides with the interval start-point. The query can then be augmented to capture this additional relationship.

queries, and integrate dependent elements such as intervals and absences. I introduce a third approach in which temporal event queries are structured as integer programs. This approach is motivated by the fact that, in certain cases, the query itself will dictate the resulting complexity, rather than the processing strategy. When a query is innately NP-Hard, there are tangible benefits that can be realized through using analogously NP-Hard processing strategies. I demonstrate this with two contributions:

- An integer programming formulation for temporal query processing.
- Quantitative evaluations of the benefits and drawbacks of this approach over sequential query processing.

A Find & Replace system for transforming event sequences:

Many visualization and statistical analysis tools allow users to hone in on critical aspects of a dataset by filtering out certain groups of records or events. For example, EventFlow allows users to filter events by category, by time, or by a given attribute. While filtering is useful for cleaving off large chunks of unnecessary data, it is not precise enough to address the common difficulties that persist in real-world datasets (described above). The following contributions address this shortcoming:

- A series of specialized data transformation controls that provide easy access to common data transformation needs.
- A Find & Replace system that allows users to flexibly and precisely alter their data (Figure 1.5).
- An integration of data transformation and visualization that allows users to see and evaluate their progress.
- A taxonomy of the various ways that a temporal event dataset may need to be altered.

- Novel metrics for evaluating the complexity of EventFlow's aggregated display (Figure 1.6).

Eight case studies that demonstrate the utility and validity of these approaches

By working with real word domain experts, I was able to observe both the limitations of EventFlow's initial implementation (without advanced query and transformation capabilities), as well as the impact that these advancements can have on the analysis process. These findings are illustrated in eight representative case studies:

- **Pediatric Trauma Procedures:** Understanding pediatric trauma team adherence to ATLS protocol.
- **LABA Guideline Adherence:** Determining whether physicians were prescribing LABAs according to FDA guidelines (Figure 1.6).
- **Opioid Use Classification:** Classifying different types of opioid use as a means of identifying patients with habit-forming use patterns.
- **Warfarin and Traumatic Brain Injuries:** Understanding the patterns of Warfarin use for patients who sustain a traumatic brain injury.
- **Radiation Classification:** Developing an algorithm capable of classifying radiation targets based on event patterns in claims data.
- **Total Parenteral Nutrition:** Determining the frequency of parenteral nutrition-associated liver disease, the face of the disease, and the severity with which it presents.
- **Trauma Role and Activity:** Understanding which tasks are typically performed during the various stages of a trauma resuscitation, in what order, and by whom.

Figure 1.5: Find & Replace provides both flexible and precise data transformation capabilities that address all three of the difficulties that arise within temporal event datasets. (Left) **The data does not match the real world events:** Disjoint intervals are merged into a single period of exposure. (Middle) **The data does not match the perspective of the study being conducted:** Visits to multiple types of doctors in close proximity are classified by the type of injury. (Right) **The data is unnecessarily cluttered:** Multiple symptoms are simplified down to only the first symptom.

- **Debugging Distributed Storage Systems:** Characterizing the behavior of a distributed storage system both during normal operation and after a crash.

1.2 Dissertation Roadmap

The rest of this dissertation is organized as follows: Chapter 2 presents a review of related work, including background material in domains such as temporal logic, temporal query specification, temporal query processing, and temporal event visualization. Then, Chapter 3 introduces the temporal entity that motivated the majority of this work, the interval, and discusses the application-level challenges of handling this event type. I then move up to user-facing challenges in Chapter 4, and the difficulties that present themselves when users specify queries involving both intervals and absences. Chapter 5 then discusses the considerations involved in processing these queries on the back-end, and presents a novel integer programming strategy for formulating temporal event queries. From there, I move on to data transformation in Chapter 6, and the various ways that a dataset can be refined to best capture the research objective. Finally, in Chapter 7, I present 8 case studies that support the utility and validity of my approaches, before concluding in Chapter 8. In this final chapter, I also discuss future directions that could be taken to expand EventFlow's capabilities in both commercial and research settings.

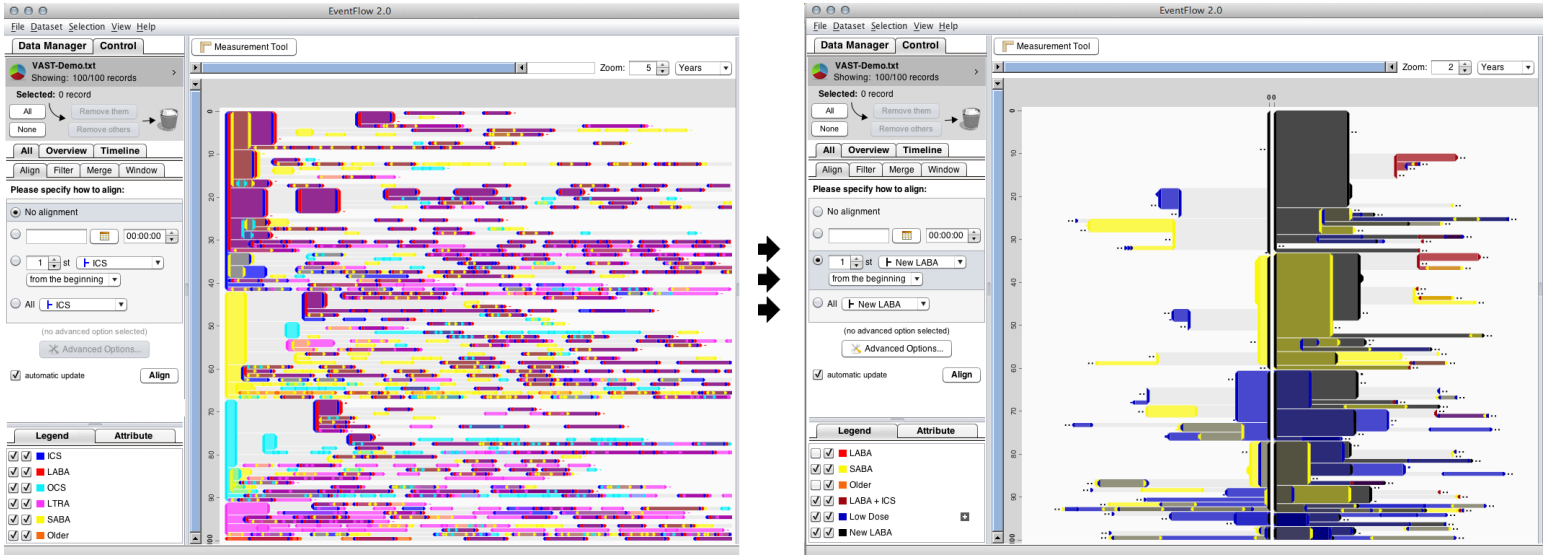


Figure 1.6: In EventFlow, the original LABA dataset, consisting of over 2700 visual elements (left), was quickly pared down to the events most critical to the study. The simplified dataset (right) consists of only 492 visual elements, an 80% reduction in visual complexity. From this simplified figure, aligned by the patients' "new" LABA prescription, researchers were immediately able to notice the data sparsity on the left side of the alignment point, indicating that patients had not received other treatments in the months leading up to their LABA prescription (i.e. not following the recommended practices).

Chapter 2

Related Work

This dissertation draws on previous work in artificial intelligence, data mining, database design, database languages, information visualization and cognitive perception. In this Chapter I present a review of this work, as well as the historical underpinnings that have brought us to this point.

2.1 Temporal Logic

At its most fundamental level, EventFlow draws on the methods to logically represent events and the relationships between them. Most notably here, is the work of Allen et al. [5, 6]. Allen identified 13 unique relationships between two intervals (Figure 2.1) and formalized a set of rules to describe them. This work has been substantially reinforced by three decades of research in the artificial intelligence community [9, 20, 55, 72, 103, 111], where Allen's relationships are used to reason about events and their impact on the surrounding world. Additional efforts in artificial intelligence and data mining research have extended Allen's work by also addressing the relationships that occur when an event sequence can consist of both point *and* interval events (see Chapter 4, Figure 4.22) [25, 72, 127]. Chen et al. also considered the scenario where point events are explicitly associated with the intervals that contain them [16].

Though Allen's 13 relationships account for every possible configuration of event end points, it has been criticized for obscuring important details within these relationships. As pointed out by [71], the three scenarios depicted in Figure 2.2 would all fall under Allen's "overlap" relationship. However, in many do-

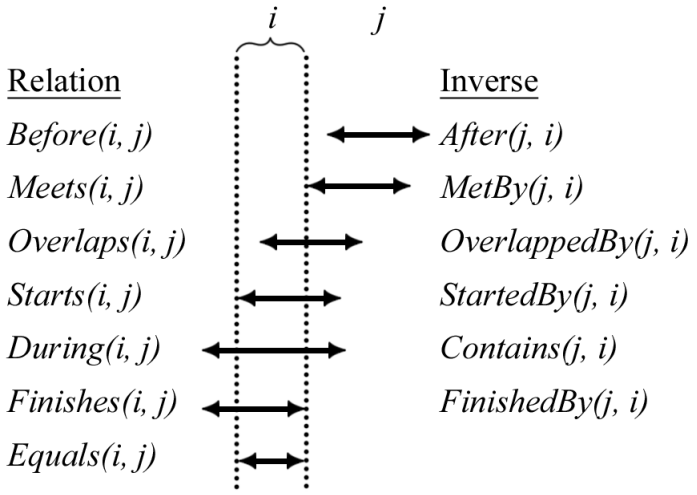


Figure 2.1: Allen's 13 interval relationships.

mains, these scenarios have drastically different implications. To this end, many researchers in the data mining community have proposed a hierarchical model of representing interval events [36, 74, 78]. Though this work is centered around automated pattern discovery, which is not a focal consideration of this work, it highlights the importance of metric relationships between events, as well as logical ones. This dual consideration was formalized by Kautz et al. to better incorporate metric qualifiers into logical reasoning [47].

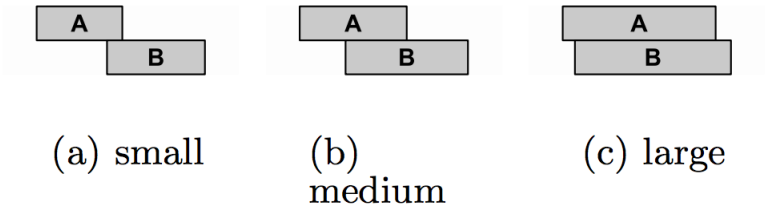


Figure 2.2: All three of the above scenarios fall under Allen's "overlap" relationship, despite drastic differences in the duration of the overlap.

Though numerous lessons can be extracted from this work, temporal logic stems primarily from the field of artificial intelligence and reasoning. In many ways, artificial reasoning can be seen as having the same trajectory as this work, but

moving in the opposite direction. Temporal logic in artificial intelligence strives to form human-like reasoning about temporal event sequences. By contrast, this work, which focuses on temporal query, strives to translate human reasoning about temporal phenomena into meaningful computational commands. That is to say, the translation is going in reverse. Despite the substantial barriers and difficulties that present themselves in this latter consideration, there has been surprisingly little work done to address this process. Chapter 4 provides initial insights into the human-based usability challenges of specifying accurate temporal event queries over complex phenomena such as intervals and absences.

2.2 Temporal Query

To understand EventFlow in the broader scope of temporal query, it is important first to understand a bit of history. Temporal query began with the need to track changes in standard relational databases. The most frequently cited example is that of a human resources database. In such a database, information would be stored for all employees, such as the department that they worked in and the supervisor that they reported to. But this information, of course, was subject to change: an employee could be switched from one department to another, or a new supervisor could be hired. These "events" needed to be incorporated into the database without losing the previous history. For example, if Jane was moved from the Finance department to the Accounting department on January 5th 1987, the database still needed to contain the information that Jane worked in the Finance department from March 3rd 1985 to January 5th 1987. Databases capable of tracking this information, were called temporal databases or transaction databases.

What's critical here, however, is the nature of the event itself. Moving an employee from one department to another is a change that would be both instigated and recorded by a human. As such, these changes - these events - occurred relatively infrequently. One might imagine a medium-sized company making only

a handful of staff changes every month. Because of this, the design of temporal databases, and the query languages developed for them, focused less on the events that incited changes, and more on the state of the database between these changes. Even early work in event calculus focused heavily on understanding the state-based intervals of time between events [53]. A query might ask, "Who was in charge of the Accounting department last March?" or "How long does an employee typically stay in any one department?" The events are necessary for answering these queries, but they are not the foremost focus.

This is where the shift has occurred. In the past four decades, our ability and capacity to record events has increased dramatically. A shipping facility, for example, might automatically log events down to the millisecond on thousands of packages going out the door every day. As a result, a second need has arisen, a need for event-centric queries. These queries search for an event pattern of interest, with little focus given to the state of the entity between events. This is the realm in which EventFlow lies. In this section, I describe how both state-centric and event-centric query systems have influenced this work.

2.2.1 The Standard Search Model

In discussing and comparing related work in both state-centric query systems and event-centric query systems, the first step is to frame temporal query in the context of a standard search model. Hearst [38] breaks the search process down into three stages: Specification, Presentation of Results, and Reformulation. For each of these stages, I discuss design goals as they apply to EventFlow and temporal query in general:

Specification: In the Specification stage, users construct and submit their initial query to the search application. Perhaps most important in this stage, is that a diverse array of potential users be able to complete the specification process. Users must be able to specify their query using a language that is both accessible and

intuitive. Furthermore, the interface should provide users with a clear idea of how the application is going to interpret their query. Errors should only arise though the users' misinterpretation of their dataset or temporal relationships in general, not through the users' misinterpretation of the query interface.

Presentation of Results: In the Presentation stage, the application returns a set of results, based on the query. The critical component of this stage is a presentation that gives users enough information to refine their query if necessary. Clinical researchers typically group retrieval errors into two categories: false positives and false negatives. An ideal interface should provide insight into both of these types of errors. Users should be able to look at the result set and quickly determine whether their query needs to be refined and if so, how. While this is a critical stage of Hearst's search model, and has been the sole focus of other work on search [14, 15, 85], the result presentation is often ignored by temporal query system designers.

Reformulation: In the final stage of the search process, users reformulate and resubmit their query based on the information obtained in the Presentation stage. Here, it is important that the interface allows users to focus solely on the refinement. Users should not have to respecify their entire query; they should be able to modify only the components that need to be either added or removed. This allows users to effectively bookmark their thought process and continually move closer to their final result set.

2.2.2 Command-Based Query Languages

Command-based query languages are the most direct bloodline back to state-centric query systems, as well as the most pervasive systems in practice. Database researchers have produced numerous works on designing algebras and languages to support access to temporal relationships. Snodgrass [100], Jensen et al. [42],

and Das et al. [24] have all proposed extensions to the well-known database querying language, SQL, to encompass temporal relationships, including constructs for specifying both intervals and the absence of an event [29]. Chomicki surveyed these works, distinguishing between the choice of temporal domain, abstract vs. concrete temporal databases, and the semantics, expressiveness, and implementation of the language used [18]. Toman further addressed the need to reconcile differences between point-based and interval-based query languages in temporal databases [107].

However, the majority of these languages and algebras were designed for state-centric systems and relational models, and have to be cleverly adapted by users to be used for event-centric pattern queries. The result is that they have a notoriously steep learning curve (see Figure 4.16 in Chapter 4), and are difficult to retain for intermittent users. This creates a substantial barrier at the Specification stage of the search process. Researchers outside of the database community cannot be expected to learn, and thus utilize these complex, command-based languages. Even for experts in this domain, command-based temporal queries can present significant challenges. The command-based specifications of temporal relationships do not match the way that people naturally think about events occurring in time. Consider, for example, a simple query for records in which Event A occurs at least 3 days after Event B. The command-based specification would likely include a code snippet that looked much like the following:

Event B + 3 days $\leq$ Event A

A small lapse in focus while specifying this code can easily result in a query where the 3 days are added (or subtracted) from the wrong side of the equation, or the equality sign is mistakenly flipped. These simple keystroke errors are both difficult to visually detect the specification and can substantially affect the query results. Figures 2.3, 2.4, and 2.5 illustrate the complexity, as well as the variety,

of command-based queries for temporal event patterns. Furthermore, command-based systems almost universally ignore the Presentation stage of the search process. Even work that explicitly attempts to bridge the gap between state-centric and event-centric systems [54, 92, 128] have a high barrier to entry due to the lack of an intuitive result presentation. In many cases, the user is simply notified that the query completed and given a count of the records selected. Because of this, it is extremely easy for mistakes in the specification stage to go unnoticed. The combination of a challenging specification process and a result presentation that does not help users recover from errors makes command-based query an untenable solution for widespread use.

```
select distinct t1.patid,
       t1.drug,
       t1.dispense_date,
       t1.next_drug,
       t1.next_dispense_date
from (select distinct patid,
       dispense_date,
       lead(dispense_date,1) over (order by patid, dispense_date, drug
          ) next_dispense_date,
       drug,
       lead(drug,1) over (order by patid, dispense_date, drug)
          next_drug
from  DRUG_TBL
where drug in('DRUG A', 'DRUG B')) t1,
       EVENT t2
where t1.patid = t2.patid and
       t2.ICD9 = 'STROKE'      and
  ((t1.drug = 'DRUG A' and t1.next_drug = 'DRUG B') and
          (t1.dispense_date = t1.next_dispense_date) and
          (t1.next_dispense_date < t2.event_start or  t1.
             dispense_date > t2.event_end));
```

Figure 2.3: Command-based specification for patients that did not have a Stroke while taking Drug A and Drug B.

```
SELECT A.Symbol, A.Tstamp, A.Price, max(B.Tstamp), min(B.Price),
FROM Ticker
AS PATTERN (A B+ E* F+)
PARTITION BY Symbol ORDER BY Tstamp
WHERE B.price < PREV(B.price)
  AND E.Price >= PREV(E.Price)
  AND E.Price < A.Price
  AND F.Price >= PREV(F.price)
  AND F.price >= A.price
```

Figure 2.4: Command-based specification of falling stock prices, followed by a subsequent price rise [128].

```
range of f1 is Faculty
range of f2 is Faculty
range of a is Associates
retrieve into Starsof1984 (Name = f1.Name, validfrom = f1.validfrom,
  validto = f2.validfrom, transactionstart = 123, transactionstop =
    0)
where f1.Name = f2.Name and f1.Rank = Assistant and f2.Rank =
    Full
  and f1.validfrom I a.validto and a.validfrom I f1.validto
  and f2.validfrom I a.validto and a.validfrom 5 f2.validto
  and 1020 I f Ltransactionstop and f1.transactionstart I 1020
  and 1020I f2.transactionstop and f2.transactionstart I 1020 and
    1020 5 a.transactionstop
  and a.transactionstart I 1020
```

Figure 2.5: Command-based specification over a state-centric database of faculty assignments [100].

2.2.3 Graphic-Based Query Languages

Just as the graphical user interface revolutionized the way that people interact with computers, visual query languages reduce complexity by conveying information with meaningful graphical representations [22,59,61]. In fact, a large body of work has focused solely on graphic-based systems for event-centric temporal querying [17, 19, 32, 35, 40, 41, 115, 121, 124]. These languages are far more learnable and usable than their command-based counterparts because they more naturally fit the way that people think about events occurring in time. Graphically, time forms a very intuitive, horizontal axis on which users can plot event patterns. Consider

the same query from the previous section (all records in which Event A occurs at least 3 days after Event B), specified using a graphical representation (Figure 2.6).

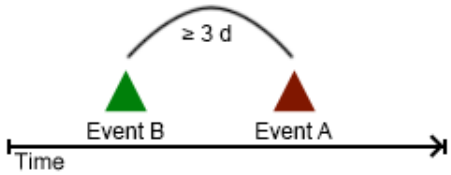


Figure 2.6: Graphically plotting events on a horizontal time axis better represents the way that people naturally think about events occurring in time.

While graphic-based languages are far more learnable and usable than their command-based counterparts, they are consistently limited in terms of query scope. None of the surveyed works offered support for both intervals and absences. Figures 2.8, 2.9, and 2.10 provide examples of graphic-based languages that support only point-based events. Other graphic-based languages were even narrower in scope: Figure 2.7 depicts a query language developed by Certo et al., which is based on the graphics used to represent finite state automaton [13]. This system, however, revolves around the metric relationship of events to absolute times and dates. A typical query in this system might be, "Find all the events that occurred every other Monday in April." Other systems, such as [84] and [26], focus heavily on the detailed specification of event attributes such as a medication dosage or attending physician. By contrast, the EventFlow visualization is designed primarily around logic-based event sequences, rather than metric qualifiers or event attributes, and should thus have a query system that reflects that focus.

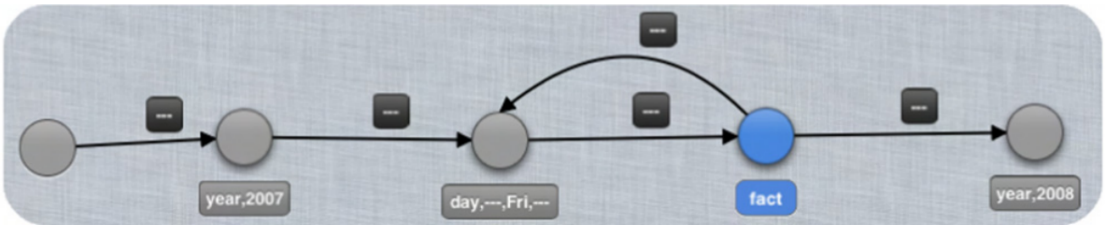


Figure 2.7: Some graphic-based temporal query languages, such as the one pictured above, focus on the relationship between events and absolute dates and times.

Due to the limited scope of graphic-based temporal query systems, visual

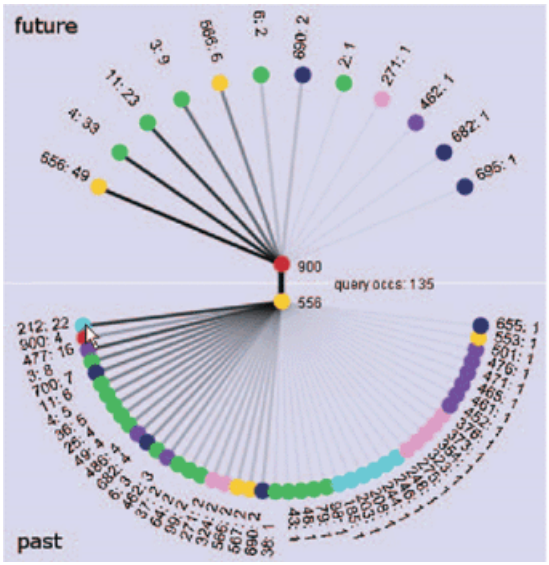


Figure 2.8: The ActiviTree query system [115] allows users to chain together sequences of events, and then shows the most common outcomes on either side.

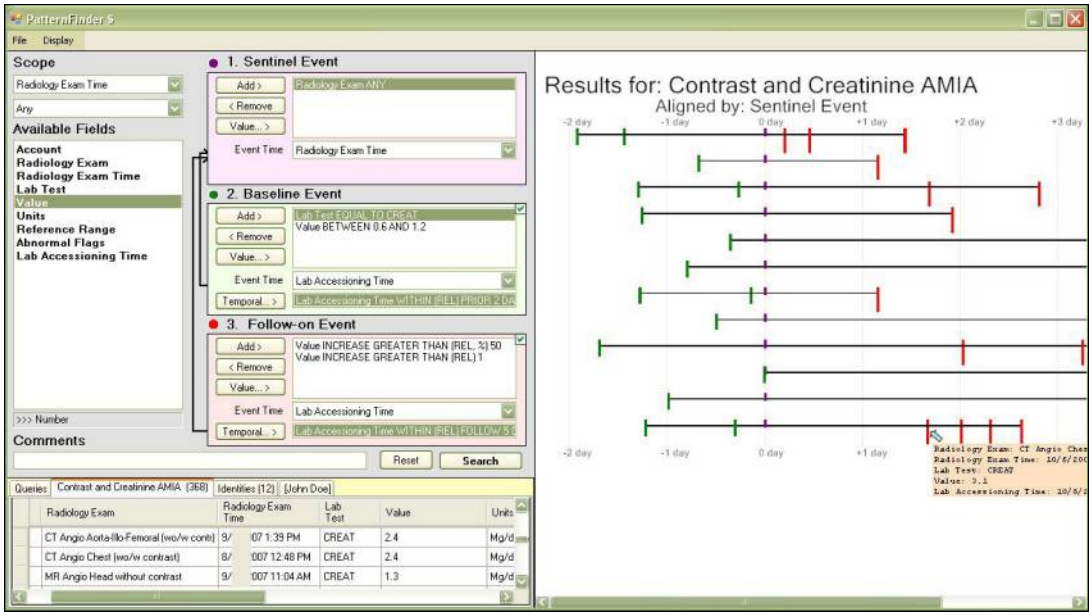


Figure 2.9: PatternFinder [82] allows users to specify complex sequence of points events using a menu-based specification interface, and integrated visualizations.

query languages in the geo-spatial domain are a relevant consideration. Analogous concepts such as “overlap” and “not-overlap” are principal tenets in geo-

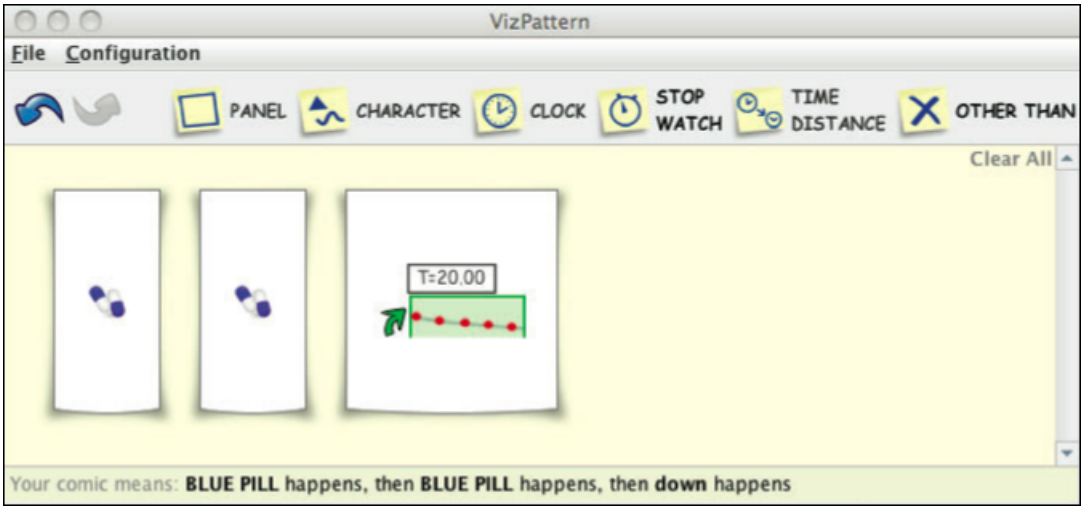


Figure 2.10: Jin and Szekely use a comic strip metaphor in their point-based temporal query system [43].

spatial querying, and user behavior in specifying these relationships is better documented [33, 87, 94]. However, the need for a second (and sometimes a third) dimension makes it difficult to draw direct comparisons between temporal event queries, and the geo-spatial domain.

Three guiding lessons can be extracted from the previous work in graphic-based query languages. The first is that the Specification stage is greatly facilitated when the graphical elements used in the query language closely resemble those that comprise the actual artifacts. For example, Jin and Szekely [43] allow users to drag elements directly from the display of patient records into the query interface. Second, users should be able to directly manipulate query elements in the display, exemplified nicely by the QueryLines system [91]. Many visual query languages use only form-based tools to modify the elements in the query display, which forces users to substantially backtrack when only minor changes are necessary. Finally, while visual query languages can make the Specification stage of the search process more accessible, they cannot single-handedly improve the Presentation or Reformulation stages. These languages must be accompanied by thoughtful strategies for result analysis.

2.2.4 Similarity Measures

Similarity measures can play a substantial role in the effectiveness of a query system. When users are presented with their search results, they must be able to detect the presence of both false positives and false negatives. An intuitive way to accomplish this is to rank both matches and non-matches by how similar they were to the target sequence. In this scheme, likely false positive would appear towards the bottom of the list of matches, and likely false negative would appear towards the top of the list of non-matching. An effective similarity measure is critical for establishing this ranking.

Temporal, categorical event sequence similarity is the comparatively less-researched combination of two very thoroughly researched domains: numerical time series similarity and string similarity. Numerical time series similarity metrics, surveyed by Ding et al. [27] and Saeed and Mark [93], are well-suited for capturing the temporal aspects of similarity, but do not address categorical event data. Conversely, string similarity metrics, surveyed by Navarro [73], are inherently categorical, but do not address temporal aspects such as time lapses and durations.

Wongsuphasawat et al. directly addressed temporal, categorical event sequences by introducing the "Match & Mis-Match" measure for computing similarity [122, 124]. This measure, however, relies on a dynamic programming algorithm that cannot backtrack, making it difficult to incorporate temporal relationships such as time-lapse constraints, absences, and intervals. Vrotsou identified a set of appropriate similarity metrics [113, 114] for categorical event data, but limited her scope to only abutting intervals. Other approaches, such as Obweger et al. and Mannila et al. [67, 76], employ more complex similarity measures, but only consider point-based events. Finally, the data mining community has produced multiple works on determining patient similarity [105, 106, 125], but has focused primarily on automated clustering, rather than ranking against a predetermined query.

2.2.5 Query Processing

Querying over temporal event data has been addressed most prominently in two primary domains: temporal database querying and data mining. The former has focused its efforts on tailoring standard, relational query languages to better account for temporal data [29,52,70,101,102]. The temporal database field, however, revolves around a state-based perspective of the database rather than an event-based perspective. That is, the foremost concern is the state of the database at any given moment, rather than the sequence of events that brought the database to that state. These adapted languages reflect this, and thus are not ideal for event sequence queries.

Furthermore, temporal databases are still rooted in the relational data model. Query languages such as TSQL and TQuel [42, 100] have constructs for specifying temporal relationships, but they function primarily as shortcuts. The query processing remains tethered to the standard relational operators. For many of these languages as well, there is not much clarity as to what should be expected when multiple event sequences within a single record can be matched to a given query. The notion of finding the best match both within a single record, and across multiple record is largely unexplored.

The data mining community has also produced a large body of work on temporal event analysis [27, 44, 57, 78, 79, 126], summarized in the survey by Laxman et al. [56]. This work, however, is heavily focused on pattern discovery, rather than query. While similarities can be drawn between these two objectives, they are not inherently the same task. Data mining, for example, typically strives to discover the longest possible significant pattern. By contrast, effective querying can be accomplished by specifying multiple small query patterns. This work draws more closely on work from the data mining community that specifically addresses query processing, such as [58], [1] and [117].

2.3 Temporal Visualization

EventFlow has evolved from a succession of increasingly powerful tools, developed by the HCIL at the University of Maryland (Table 2.1).

Tool	Event Types	Records	Display
LifeLines [83]	points, intervals	one	individual
LifeLines2 [116]	points	many	individual
Similan [124]	points	many	individual
LifeFlow [123]	points	many	individual, aggregated
EventFlow	**points, intervals**	**many**	**individual, aggregated**

Table 2.1: A summary of the HCIL's evolving work with point and interval event types, one and many patient records, and individual and aggregated record displays.

EventFlow also draws on other work in the field of visualization, and the varied attempts to visualize different kinds of temporal data. Aigner et al. took a comprehensive look at such attempts, covering the differences in both data types and visualization techniques [2]. Rind et al. narrows this focus considerably in a review of visualization for medical data, specifically electronic health records [88]. None of the implementations surveyed in either of these works, however, incorporate both point and interval events into an aggregated display. Figures 2.11 and 2.12 depict examples of aggregated visualization that handle only point-based event.

2.4 Display Simplification

When datasets consist of thousands of records, each with fifty or more events it becomes difficult for visualizations to provide intuitive, overview displays, while still rendering every event. Attempts to create simpler displays thus far have revolved primarily around one or more of the following four filtering strategies:

- Removing a subset of the records from the display.
- Removing a subset of the event categories from the display.

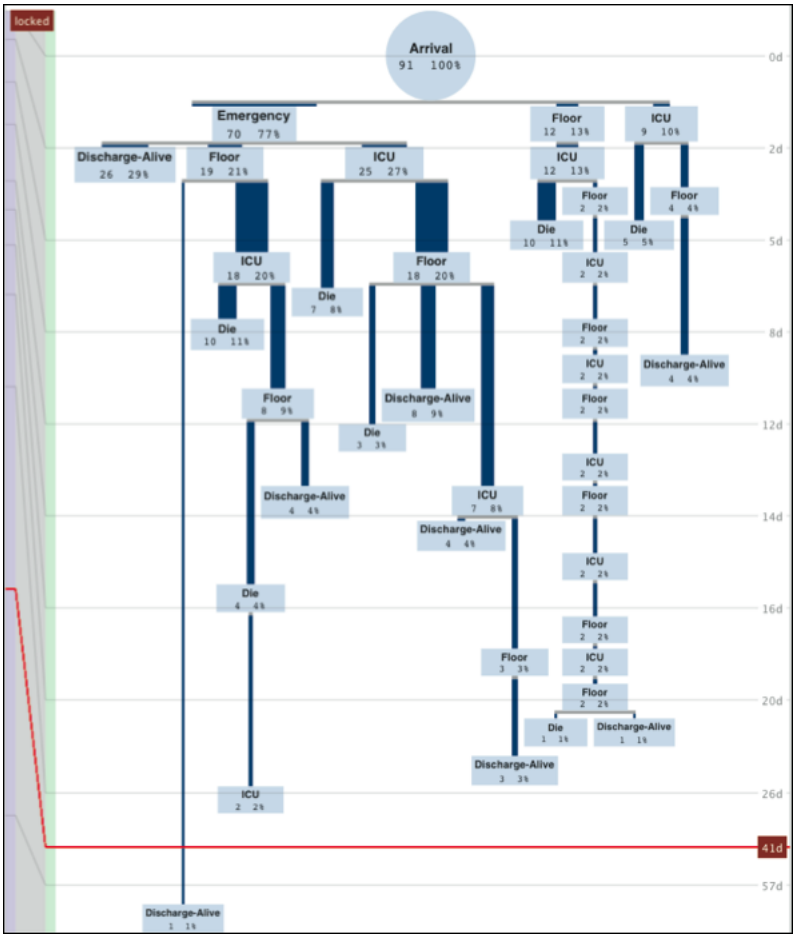


Figure 2.11: VisCareTrails [62] tracks event sequences using a timed word tree visualization.

Figure 2.12: The eSeeTrack system [108] uses a tree-based visualization to depict common sequences of fixation gleaned from eye tracking software.

- Removing a section of the time spectrum from the display

- Filtering out a subset of events, based on the event attributes.

EventFlow currently provides users with all four of these filtering options. However, in many cases they remove either too much or too little information. Users need more fine-grained control over which events will remain in the display, and how to account for the events that are being removed. Alternate approaches create simplified displays by automatically abstracting various aspects of the dataset [3]. In these cases, users are given a set of options that dictate how the dataset is abstracted, but do not have control of the display beyond that. There are currently no visualization tools that put event sequence transformation directly in the hands of the users.

However, this work can draw on simplification strategies from other domains of data visualization. For example, Dunne et al. proposed the simplification of network graphs using glyphs to represent common network structures [30]. I also draw on general work on the detection and measurement of visual clutter [90]. Rosenholtz et al. correlated search performance in complex imagery with various measures of visual clutter including Feature Congestion [89], Subband Entropy [65], and Edge Density [66]. They found, among other things that the number of objects in a display (the set size), as well as color variability are key factors in the appearance of visual clutter. These measures can be used to gauge the impact of data transformation on the resulting display.

2.5 Summary

Literature from both the artificial intelligence and data mining communities has highlighted the importance of reasoning about events that occur over a duration of time: intervals. Intervals are a fundamental temporal phenomenon from which we can discern relationships such as “during” and “overlapping.” Though efforts in database languages and data mining have made these interval relationships more computationally accessible, little work has been done on two critical fronts:

- Addressing the need to clean messy temporal data at the event sequence level. For example, temporal databases allow users to revert records back to a specific point in time, but they do not directly support large-scale changes to the events themselves.

- Building systems that help users, and in particular users who are not database experts, accurately specify and evaluate temporal event queries. For example, while Allen identified the 13 possible relationships between two interval events, he does not make any recommendation for a user-facing specification interface.

This work builds on the strong computational foundations presented here to make temporal event data and temporal event querying more accessible to a broader audience. This data type will play a central role in a data driven future, and this work offers a first look at the practical challenges and novel solutions that can be arrived at as more and more users are required to interpret this data.

Chapter 3

Interval Events: From Data Structure to Display

"It needs to be able to handle interval events."

This was the request, made of the original LifeFlow software, that started everything. LifeFlow had caught the attention of epidemiologists at the US Army Pharmacovigilance Center (PVC), who thought that the software might help them better understand prescription patterns. The catch was that LifeFlow was only set up to handle point events. That is, events that occur at a single point in time. The PVC had data that dealt exclusively with medication prescriptions, which are taken for days, or even weeks at a time. They needed to represent these prescriptions with both a start time and an end time, indicating the duration of the prescription. They needed interval events.

On the surface, the request seemed simple enough, something on the order of "I'd like fries with that" or "Yes, I'll go for the car wash." This simplicity, however, quickly dissolved into the realities of implementation and user behavior. Unlike point events, which subscribe only to the sequential notions of "before" and "after," interval events introduce the concept of "during." Take, for example, the following two event sequences:

Stroke → Admitted into Hospital → Diagnosed

Started Medication A → Stroke → Ended Medication A

While both of these sequences contain three points in time, the second sequence

conveys two additional pieces of information. First, the start of the medication necessarily implies, and is fundamentally linked to the end of that medication. Second, a stroke (a point event) occurs *during* the time that the patient was taking that medication (an interval event). This additional information changes the types of questions that users can ask of the data. It changes the way that the data must be visually displayed in order to remain intuitive, and it changes the requirements on the underlying structure in which the data is stored. Interval events represent a fundamental increase in complexity at every level of the application.

The goal of this work was to integrate interval events into LifeFlow in a way that, to every extent possible, masked this complexity from users. The software, which evolved into what is now called EventFlow, is designed to present as a simple and intuitive extension of the original LifeFlow tool. This chapter, however, details the underlying challenges and considerations that comprise EventFlow's back-end; the side of the software that the users will never see.

3.1 Input Processing

The original LifeFlow input format called for three primary columns of data:

Record ID	Event Category	Time

To incorporate interval-based events, the input format was modified to include one additional column:

Record ID	Event Category	Start Time	End Time

In this new data scheme, interval-based events are specified using both a start and end time. Point-based events are specified using only a start time, and leaving the end time column blank.

This seemingly simple addition introduced a new level of complexity in the process of reading the input file into the EventFlow data structure. Previously, the only concern was whether or not the input file was structured correctly - that is, the input file needed to have at least three entries in every row, and the third entry needed to be a time. The addition of interval-based data introduces three further, logic-based concerns that had to be addressed in order for the input file to be correctly processed into the EventFlow data structure.

3.1.1 Consistency

In order to produce a consistent set of events, it was necessary to think of each event category as either point-based or interval-based. A stroke, for example, is likely to be considered a point-based category, whereas a prescription is an interval-based category. Whether a category is point-based or interval-based changes the way that it's events are represented both in the data structure and in the display. As such, for a given input file, it is necessary for all events of the same category to remain either consistently point-based *or* consistently interval-based. For example, the following two data entries would NOT be allowed to exist in the same input file:

Record	Event Category	Start Time	End Time
106	Stroke	11/3/11	
213	Stroke	11/3/11	11/4/11

To account for this, the EventFlow input processor categorizes each category as either point-based or interval-based according to the *first* entry in which that category appears in the input file. If a subsequent entry for that category is not consistent with that original designation, users are notified of the error, and the input processing is stopped.

3.1.2 Time-Granularity

For certain datasets, it is sometimes the case that the granularity of the time entries is not fine enough to represent the interval with disparate start and end times. Take, for example, the following event entry:

Record	Event Type	Start Time	End Time
342	Therapy	11/3/11	11/3/11

In the above example, time is only specified down to the day, but the given interval takes place within a single day. The result is an identical start and end time. In instances such as this, the EventFlow input processor notifies users that an interval with the same start and end time has been found. Users then have the option to either abort the input processing, or increment the end time by one unit at the finest granularity. In EventFlow, the finest granularity is milliseconds. Thus the above entry would be adjusted to:

Record	Event Type	Start Time	End Time
342	Therapy	11/3/11 00:00:00:000	11/3/11 00:00:00:001

While this adjustment allows the underlying data structure to remain consistent, it does not address the possible effect that time granularity can have on querying. This consideration was not a pressing concern of my case study partners, and was thus determined to be outside the scope of this work. However, time granularity, as it pertains to querying, is discussed in the future work section of Chapter 8.

3.1.3 Mis-sequencing

It is also possible for an interval-based event to appear in the input file with mis-sequenced start and end times. That is, the end time occurs before the start time:

Record	Event Type	Start Time	End Time
342	Therapy	11/3/11	11/2/11

For these sequencing errors, users are notified of the error, and the input processing is stopped.

3.2 Data Structure Requirements

The data structure underlying EventFlow consists of both the individual-record structures that comprised the original LifeLines2 implementation, and the tree structure that supports the aggregation of these individual records for the LifeFlow display. EventFlow's inclusion of interval-based events required two major changes to the individual-record data structure, which are propagated up to the tree structure.

3.2.1 Event Structures

Individual records (i.e. one patient record) are maintained in two forms: as a master list of ordered events, and as a series of smaller, ordered lists, one for each event category. A full description of this data structure and the rationale behind it can be found in [116]. In this original implementation, events were not related to each other in any way apart from the order in which they are stored.

In EventFlow, a single interval event is incorporated as two point events, representing the starting point and ending point of the interval, in both the master list and the category list. While they function as point events in many regards, they have been flagged with an interval indicator, and contain a pointer to their interval "mate." The intuition should follow that interval categories will always have an even number of events.

The use of two, connected point events to represent intervals enables both the start and the end of the interval to be accessed in sequence with events of other

categories. As a result, event points that are stored in between these two indices in the master list represent the events that occurred during the interval in question.

To further facilitate access, I created a third, ordered list structure in which interval events are stored. In addition to being represented in the master list and a category list, each interval category maintains a structure in which it's events are represented as two ordered lists: a list of start events, and a list of end events. For example, an interval category A with two entries, (A_{1S}, A_{1E}) and (A_{2S}, A_{2E}),would be stored as follows:

Master List

A_{1S}	x	x	A_{1E}	x	A_{2S}	x	A_{2E}	x	x

Category List

A_{1S}	A_{1E}	A_{2S}	A_{2E}

Split Category List

A_{1S}	A_{2S}
A_{1E}	A_{2E}

3.2.2 Span Structures

The inclusion of intervals also presents the challenge of representing what is happening *between* events. Two sequential events no longer have an implied, empty space between them. It is possible that one or more intervals are spanning this space. If this information is not directly represented in the data structure, then answering a simple question such as "Which medications was the patient taking when she had a Stroke?" would require a check of each interval in the record to see if it contains that point.

While this type of question is important for querying, it is even more relevant for

EventFlow's tree structure, which underlies the aggregated display. The EventFlow display can be aligned by any point in the dataset. When this happens, two separate trees are built, one representing the events before the alignment point, and one representing the points after the alignment point. Without direct access to the open intervals between each pair of events, each event record would have to be scanned twice, once in each direction, every time the tree is rebuilt.

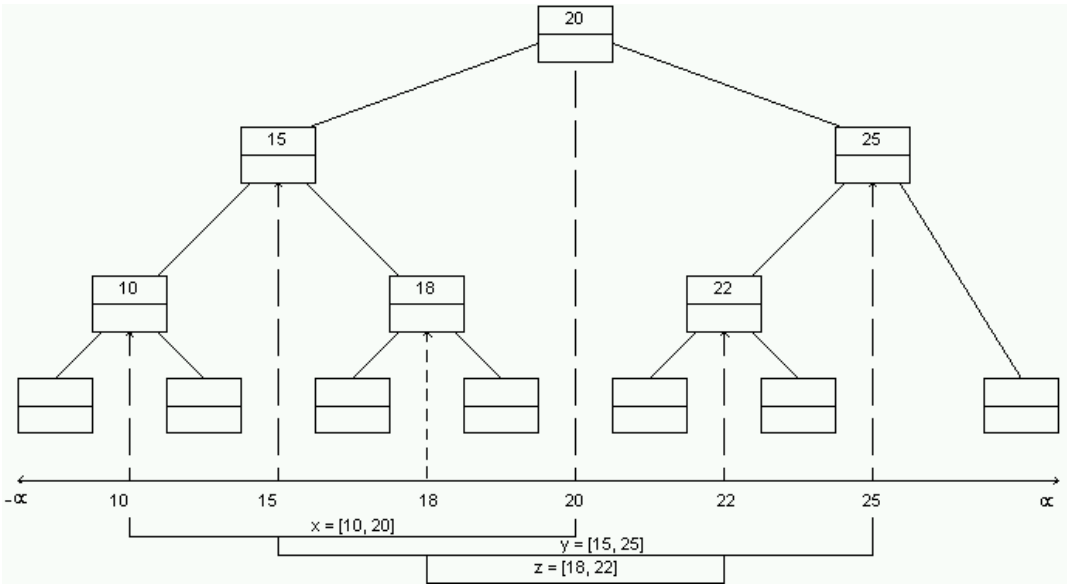


Figure 3.1: An interval tree stores a series of intervals as a balanced tree structure.

In EventFlow, the data structures that store information between events are referred to a "span" structures. There are essentially two options for designing these span structures. The first is a hierarchical structure such as a interval tree (Figure 3.1) [86]. The advantage of this structure is that it puts a strict upper bound of O(n log n) on the amount of space that the structure uses, where n is the number of event points in the record (intervals have two event points). The disadvantage is that access time of the structure is O(log n) and the tree must be rebuilt whenever the underlying event data changes (an O(n log n) operation). Thus, even the simplest operations in EventFlow, such as adding or removing a category from the display, would require the tree to be rebuilt.

By contrast, EventFlow utilizes the second option, in which each span maintains its own list of open intervals. In EventFlow, this list is indexed by category, facilitating queries such as "Was the patient taking Medication A when they had a Stroke?" The structure can be built in $O(n)$ time and accessed in constant time. When event categories are added or removed, each span structure can spawn a subset of itself in $O(c)$, where c is the number of active categories. The disadvantage of this structure is that the upper bound on space is $O(n^2)$, however, the real-world datasets we've encountered thus far do not come close to approaching this bound. Furthermore, although EventFlow currently employs custom, in-memory data structures, it may become practical to replace this with a database back-end. Were this to happen, the listed span structure can be easily transferred to a table scheme, whereas interval trees cannot.

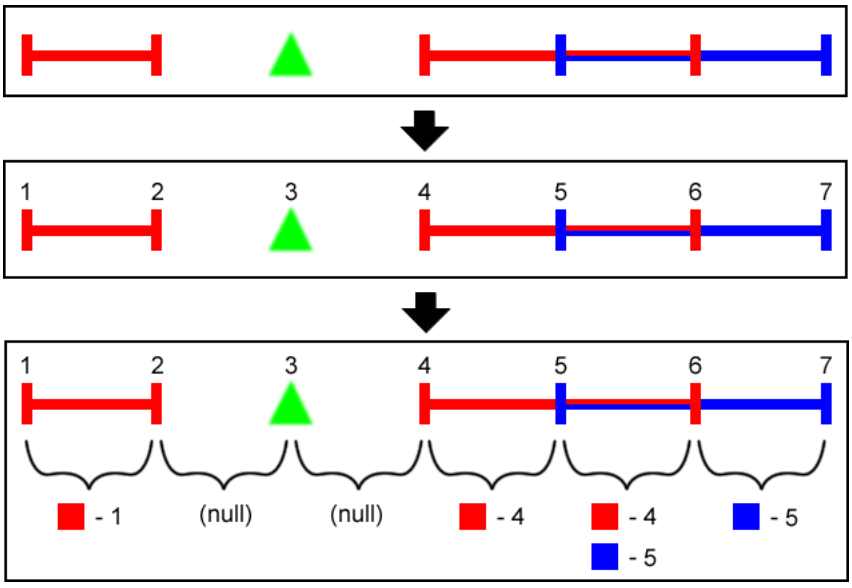


Figure 3.2: Span Structures: Between each pair of sequential events, EventFlow maintains a list—indexed by category—of the open intervals.

3.3 Display Methods

The EventFlow display consists of the individual record lists that comprised the original LifeLines2 implementation, the aggregated record display that was introduced in LifeFlow, and the controls for manipulating them. The complete view of the EventFlow interface can be found in Figure 6.1.

3.3.1 Point Event Display

Point events are represented in EventFlow as colored triangles, both in the legend (bottom left of display) and within the individual records (right pane of display). For individual records, events of the same category are drawn in the same color across a unique horizontal axis. Thus, the original sequence of three point events is drawn as follows:

Stroke → Admitted into Hospital → Diagnosed

Figure 3.3: An individual record containing only point events.

In the aggregated record display, sequential point events are displayed as a series of vertical, colored bars (middle pane of display). Thus, the aggregated display of the above sequence is drawn as follows:

Figure 3.4: The aggregated display of a sequence of point events.

The height of the bars is used to convey the number of records that consist of this identical sequence. For example, in Figure 6.1, four of the nine patient records (listed in the right pane) consist of this sequence. Accordingly, the sequence

accounts for just under half of the vertical space in the aggregated display. The space between each pair of bars is computed from the average lapse of time between those two events. A complete description of the original LifeFlow display can be found in [123].

3.3.2 Interval Event Display

To include intervals in the display, EventFlow needed to indicate not only the interval's beginning and end, but also that the event was still happening during the time in between. To do this, interval categories are represented in the EventFlow legend as a colored square (Figure 3.5). In the individual record display, this squared is split into a connected pair of bars to represent the the duration of each interval event. An event sequence involving an interval then, is drawn as follows in the individual record display:

Started Medication A → Stroke → Ended Medication A

Figure 3.5: An individual record containing point and interval events.

In the aggregated display of interval events, the interval start and end points are drawn as thin, beveled bars. The area between these bars is then filled with a less saturated version of the appropriate category color. This technique of representing intervals using filled bars has been widely used in the visualization field [50,75,109]. The aggregated interval sequence is drawn as follows:

Figure 3.6: The aggregated display of a sequence of a point event occurring during an interval event.

3.3.3 Interval Opacity

In certain instances, the sequence of event points may be more important than the duration of the interval events. In these instances, the bright color of the interval background may serve as a distraction. To reduce this visual complexity, users are able to control the opacity of the interval area (Figure 3.7). This allows the display to be quickly reduced to that of a purely point-event display when necessary.

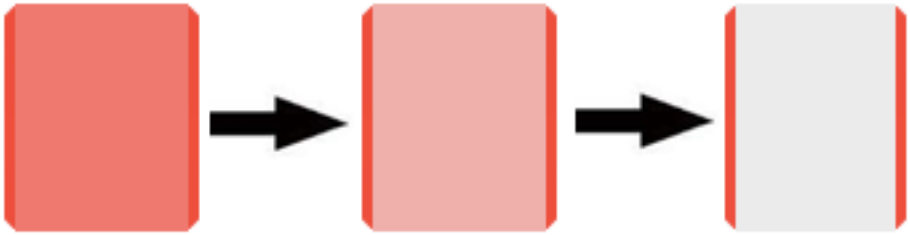


Figure 3.7: Varying levels of opacity for an interval event.

3.3.4 Overlapping

The EventFlow display must also take into account the fact that multiple interval events can occur concurrently. In all of the surveyed works, overlapping intervals are displayed by tiling them vertically [21,37,75]. For individual records, this technique is used as a natural consequence of the LineLines2 display scheme. However, in the aggregated display, vertical delineation is already used to convey diverging event sequences.

As a result, EventFlow renders overlapping intervals by filling the overlap region with the combined color of the two overlapping categories. This design decision was based on MacDonald's color guidelines [64], which recommend the use of color transparency to show overlays of related structures. The colors selected for interval categories default to primary colors, resulting in intuitive overlap colors. For example, when a red interval intersects a blue interval, the resulting overlap is purple. When two intervals of the same category intersect, the color saturation in the overlapping region is increased. The overlapping display is demonstrated

below:

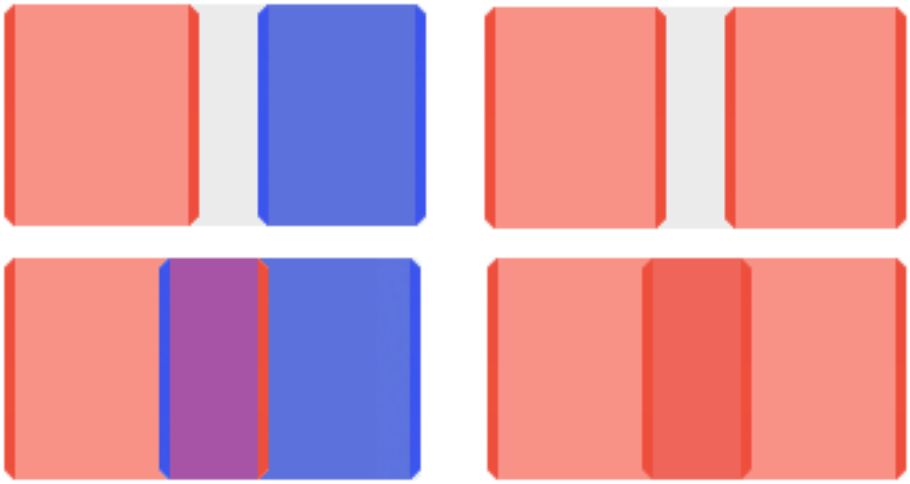


Figure 3.8: Overlapping of two interval events of different categories (left). Overlapping of two interval events of the same category (right).

This overlap display technique is effective for up to three interval categories, where an overlap between any pair of categories results in an intuitive, composite color, and an overlap of all three categories produces a less attractive, but still unique color. The motivation in allowing for this limitation stems from Shneiderman's visualization guidelines [97], which recommend providing an overview of the data, and details only on demand. As described in the previous section, the duration of an interval, and thus the fill color between its two end-points, is only important when users are addressing questions about overlapping events. Thus, when providing an overview of the data, as is the case in the aggregated display, it was important to employ a solution for overlapping events that would not visually distract from the other temporal aspects. When users do need to focus on overlapping events, these details can be brought to the forefront using one of two strategies:

1. Shift the primary color scheme to the current categories of interest.

2. Use the Highlighting feature of the Basic Search Interface (Section 4.2) to

highlight the important overlappings.

The researchers at the PVC, while frequently working with datasets that involved more than three interval categories, were typically focused on at most three categories at any single point in time. For example, they investigated the interplay between one particular asthma medication and all other asthma medications. While this dataset consisted of far more than three interval categories, for their specific purposes, all but one of them could be assigned the same color, resulting in two overall interval colors.

3.3.5 Sequencing

EventFlow's aggregated display is constructed such that an ordering must be imposed on the underlying event sequences, even when events are concurrent. To convey concurrency despite this requisite ordering, EventFlow renders concurrent events directly adjacent to one another (see Figure 3.9). Non-concurrent events are rendered with a minimum of 5 pixels in between them, regardless of scale, which allows users to easily distinguish between concurrent and non-concurrent events.

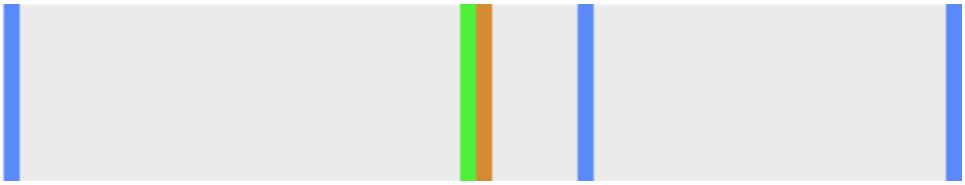


Figure 3.9: Two point events that occur at the same time are rendered directly adjacent to one another.

For concurrent point events, the imposed ordering is determined by a category ranking that is adhered to throughout the display. That is, if Stroke events are ranked above Diagnosis events, then Stroke events will be ordered before Diagnosis events in the aggregated display whenever events from these two categories occur concurrently. This ranking is determined arbitrarily when a dataset is first loaded, however, users can customize the ranking by re-ordering the event categories in the legend.

Concurrent interval events require a more involved ordering solution because ordering can affect the way that users interpret the event relationships. Consider, for example, the scenario where a Stroke event occurs exactly when one interval ends and another interval begins (Figure 3.10). Does the Stroke occur during the Drug A interval? Does it occur during the Drug B interval? Does the Drug A interval overlap the Drug B interval?

Figure 3.10: A point event occurs exactly when one interval ends and another begins.

Strangely, the answers to these questions seem to be: yes, yes, and no. Users, when presented with these scenarios individually, usually agree that the Stroke event happens during an interval if it is concurrent with either the interval start or end point. However, they do not consistently agree on whether an interval that starts exactly when another interval ends is overlapping that interval. Section 3.4 provides some insight on why this might be the case. Hopefully it is clear, however, from a display perspective, that there is no logical way to represent the Stroke event happening during both intervals without the intervals overlapping.

This inconsistently leaves EventFlow, and thus the users, with two display options: fully-overlapping, and non-overlapping. In the fully-overlapping mode, concurrent event points are rendered to convey the maximum possible overlapping. In the non-overlapping mode, concurrent event points are rendered to convey the minimum possible overlapping. The sequence depicted in Figure 3.10 is shown in both fully-overlapping mode and non-overlapping mode in Figures 3.11 and 3.12 respectively.

3.4 State-Based Intervals

As described in Chapter 2, there are 13 different ways in which two intervals can occur in relation to one another. There are 5 different ways that a point event

Figure 3.11: Fully-Overlapping: A point event occurs exactly when one interval ends and another begins.

Figure 3.12: Non-Overlapping: A point event occurs exactly when one interval ends and another begins.

can occur in relation to an interval, and there are 3 different ways that a point event can occur in relation to another point event. Despite this complexity, real-world datasets rarely contain all 21 of the possible temporal relationships between two events. This section details the common scenarios in which datasets can be expected to contain only a subset of these relationships. They offer opportunities to selectively relax or tighten components of the data structure and processing.

Consider a patient who arrives at the hospital, and is checked into the Emergency Room. There, the patient receives several treatments over the course of an hour before being transferred to the Intensive Care Unit. Additional treatments are administered there throughout the next few days. The patient spends periods of time connected to an assisted breathing machine, eventually recovering the ability breathe without it. At this point the patient is again transferred, this time to a Floor room, and finally discharged. What can be inferred about the temporal relationships in this patient record?

3.4.1 Record-Level States

In order to ask questions like, "Which treatments did the patient receive while in the Emergency Room?" it is convenient to store the patient's stay in each department as an interval event. When the patient is transferred to a new department, the interval for the current department ends and the interval for the new department begins. This is referred to as a record-level state transition. The end of one interval necessarily implies the beginning of another. And since it is impossible for a patient to be in two departments at the same time, these intervals will never overlap.

This is a common scenario in temporal event datasets across a variety of domains. For example, the basketball datasets discussed in Chapter 6 adhere to a record-level, state-based scheme. Record-level states reduce the number of possible interval relationships from 13 to only 2 (see Figure 3.13). They also significantly reduce the size and complexity of the information that must be stored in the span structures. Instead of maintaining indexed lists of open intervals, or building time-intensive tree structures, each event need only point to the single event category it is contained by. This adjustment can significantly reduce the overall memory load. Lastly, record-level states reduce the complexity of the resulting display, as there is no need to account for interval overlapping. Visualization systems like [10] are designed entirely around this assumption.

3.4.2 Category-Level States

Returning to the initial example of a patient being treated throughout the various departments of a hospital, let us assume that one event category in this dataset is an interval category that represents when the patient is connected to an assisted breathing machine. Intuitively, at any given time, the patient is either connected to the breathing machine, or not connected to the breathing machine. The patient will never be connected to two breathing machines and, as a result, the events of

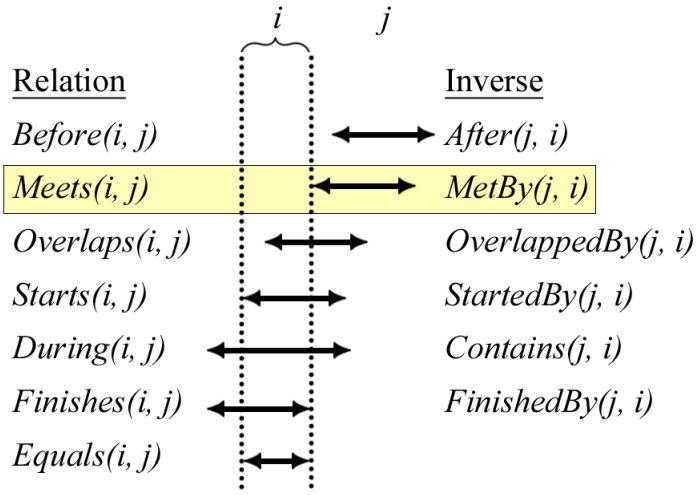


Figure 3.13: Record level states limit Allen's 13 interval relationships to only 2.

this category will never overlap. Interval event categories for which no two events of that category can overlap are referred to a category-level states.

Category-level states simplify the requirements on the underlying data structure in that the amount of information that can be stored in each span structure is bounded by the number of categories, as opposed to the number of events in the record. For long records with only a small number of categories, this can be a substantial reduction in complexity.

3.5 Summary

Intervals are fundamental in representing real-life events that occur over a duration of time. On the surface, intervals can present as a straightforward extension to point-based events, requiring only that two points in time be uniquely linked. However, the temporal relationships that these links create, and their varied implications create a complexity that touches every aspect of a temporal analytics application. In this chapter, I discussed the considerations that must be taken in the input processing, the data structure, and the display in order to effectively handle interval events.

While these considerations are critical in implementing an effective temporal analysis tool, they remain largely invisible to users. In the following chapter, I present an in-depth review of how users reason about intervals, and how they construct queries to find the unique relationships that interval events create.

Chapter 4

The Usability Challenges of Temporal Query

In 1992, a small class of elementary school students (including the author of this paper) was given a simple writing assignment: describe how to make a peanut butter and jelly sandwich. After a short period of fervent scribbling, papers were handed in to a teacher, who sat at the front of the room, surrounded by every tool and ingredient that one might need to construct the aforementioned confection. Then, one by one, the teacher acted out each set of instructions with the strictest adherence, and with total disregard for disaster or absurdity. Amidst the cackling laughter of the delighted students, peanut butter was spread onto walls, jelly onto desks, and bread was pressed into the carpet. The lesson: complexity can mask itself within a seemingly simple concept.

Nearly 20 years later, I found myself in the same position as I watched users specify temporal queries: they would begin with an idea that was so simple and purposeful, and end up with metaphorical peanut butter smeared onto metaphorical walls.

The difficulty with questions involving temporal event sequences is not necessarily understanding the underlying complexity, but articulating that understanding into meaningful queries. Users assume that the simplicity with which they communicate about these events in plain English will translate just as easily to the underlying search application. For queries involving patterns of point events (events that occur at a single point in time), this assumption typically holds true. However, such simple events do not adequately cover the complete range of both medical and real world phenomena. My primary collaborators, epidemiologists at the US Army Pharmacovigilance Center (PVC), are primarily responsible for

conducting drug related studies involving prescription administration and medication interaction. Their data and their inquiries, are inherently interval-based. For example, they might need to know when two medications are being taken at the same time. Additionally, they frequently explore questions in which the absence of an event is the critical point of interest. For example, they might be looking for patients who did *not* experience a symptom after receiving a medication.

When queries involving intervals and absences arise, our natural language for describing event relationships completely masks the underlying computational complexity. The result is that users specify queries that fail to match their intention. This problem occurs across a wide range of both clinical and non-clinical domains.

In this chapter, I present a two-part user study, designed to determine the central difficulties of temporal query specification, and alleviate these difficulties through graphical interaction. I report on the design of a basic, menu-based interface and an advanced, graphic-based interface, which are integrated into EventFlow's control panel and search tab respectively. I then assess the performance of these query interfaces in helping the PVC epidemiologists answer their evolving research questions.

4.1 The Challenges of Intervals and Absences

Reasoning about intervals and absences is central to clinical research as well as many other domains of temporal search. As such, my first goal was to better understand the types of questions that users ask of their data, and the difficulties they encounter when trying to formulate those questions into meaningful queries.

An understanding of temporal query construction was initially shaped by a series of tests and interviews with users of an early version of the EventFlow visualization tool. I conducted both individual and group user sessions with seven researchers at the PVC, for a total of eight hours. In addition, I conducted similar

one-hour sessions with many other clinical researchers, including researchers at Washington DC Children's Hospital, Yale Medical School, and Harvard Medical School. From these sessions, five common difficulties began to emerge across users from multiple clinical domains.

(The graphics used in Figures 4.4 through 4.15, while rooted in those used in the advanced query interface, are adapted here to illustrate the difficulties being described.)

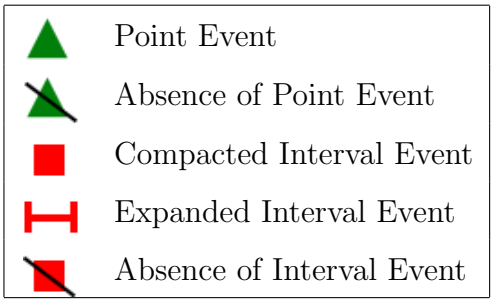


Table 4.1: Event graphics, for reference throughout this section.

4.1.1 Difficulty 1: Intervals and the Elusive "OR"

One of the largest obstacles in constructing effective temporal queries is the English language. Our natural language for describing interval event relationships completely masks the underlying computational complexity. Specifically, with queries involving intervals, English allows users to silently communicate the word "or." To illustrate this oddity, consider the following query involving two point-based events:

Find patients who had a Stroke after visiting the Emergency Room.

The graphical specification of this query comes as no surprise, as shown in Figure 4.1.

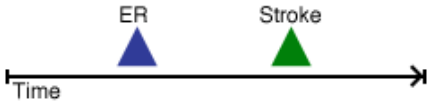


Figure 4.1: Patients who had a Stroke after visiting the ER.

Now consider that this query needs to be augmented to also capture the reverse relationship:

Find patients who either had a Stroke after visiting the Emergency Room, OR had a Stroke after visiting the Emergency Room.

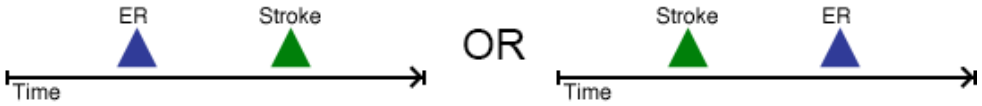


Figure 4.2: Patients who had a Stroke after visiting the ER, or had a Stroke before visiting the ER.

Much like the plain English description, the graphical specification is augmented with this additional sequence (see Figure 4.2). With point-based events, users benefit from this one-to-one relationship between the complexity of the plain English description of the query, and the complexity of the actual query specification. This allows them to specify queries much in the same way that they naturally think about them. Now consider the following query involving two interval events:

Find patients who took Drug A and Drug B at the same time.

This is one of the simplest and most common queries that a user might have of a dataset involving interval events. However, when translated into the actual query that must be computationally processed, this query actually represents nine different event relationships that must be found (see Figure 4.3). This disconnect between the complexity of the plain English description and the complexity of the actual query specification makes it extremely difficult for users to make this translation. I observed that users frequently know that multiple relationships must

be captured, but rarely are able to conceive of all the possible relationships that capture their plain English description. Furthermore, this example only involves two intervals. When additional intervals are involved, the complexity disconnect is even more dramatic.

The lesson to be learned here is that users are very likely to miss relationships that they are in fact, trying to capture. As a result, an effective query interface should be able to highlight false negatives. Users must be able to notice these relationships that they missed, and have some way of incorporating them into their result set after the execution of their initial query.

4.1.2 Difficulty 2: Specifying Intervals as Point Events

As mentioned previously, most users are able to easily understand temporal relationships involving point events, and specify successful queries to access them. What I frequently saw then, was that users would attempt to apply these same querying strategies to questions involving intervals. That is, they would specify a query for an interval event as if it were two separate point events (one representing the interval's start and one representing the interval's end). Thus, they would try to specify the following event sequence, where a Stroke occurs *while* the patient is taking Drug A (Figure 4.4).

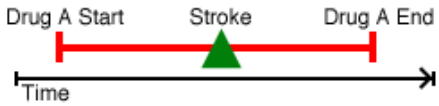


Figure 4.4: Intended query: Stroke occurs during Drug A.

Unfortunately, they would construct the query as three disjoint event points - paraphrased as:

Find patients for whom Drug A was started, who then had a Stroke, and who then stopped taking Drug A.

This query is analogous to the query in Figure 4.5:

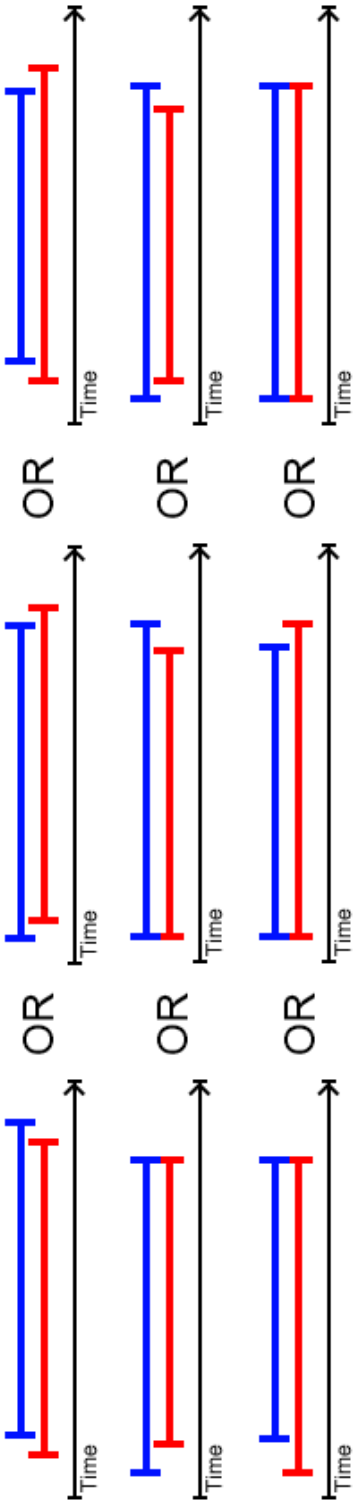


Figure 4.3: Patients who who took Drug A and Drug B at the same time.

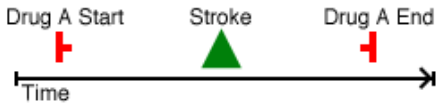


Figure 4.5: User specified query for a Stroke occurring during Drug A.

The problem is that users perceive the beginning of the Drug A prescription as a context change: the patient is now taking Drug A. They assume that this context remains in place until the end of Drug A is specified. They also assume that the underlying application will interpret this context in the same way, however this is not necessarily the case. Many databases and data structures store interval events as two separate point events, linked by a unique identifier. Thus, this query returns cases like Figure 4.6, where a Stroke occurs between two separate prescriptions.

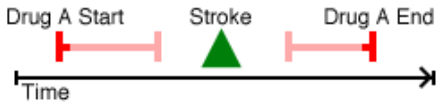


Figure 4.6: False positive for "during" query.

The lesson here is that the fundamental temporal relationship created by intervals, that of "during," cannot be adequately specified using the same "before" and "after" strategies used to specify point event relationships. Interestingly, this issue only arose (but arose very frequently) during my initial tests with clinicians, who primarily used command-based querying tools. The graphical representation of intervals that was used throughout the design study completely eliminated this problem. This shows that a successful query tool can guide users away from these errors by explicitly linking interval end-points in the query specification.

4.1.3 Difficulty 3: Specifying Absence as Non-Presence

Throughout my interviews, users often assumed that by not specifying the presence of an event, they were, by implication, specifying its absence. For example, a researcher looking for potential causal relationships may need to find all patients who were prescribed Drug A, and then had a Stroke after the prescription ended.

The resulting query would frequently look something like Figure 4.7:

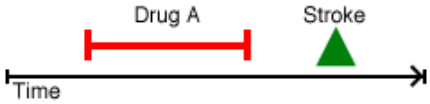


Figure 4.7: User assumes an implied absence of a Stroke during Drug A.

The assumption here, is that the absence of a Stroke while taking Drug A is implied by the fact that the presence of a Stroke was only specified after the Drug A interval. The problem with this assumption, is that it also necessarily implies that no other type of event occurred while taking Drug A. It also then implies that no events occurred before Drug A, or between Drug A and the Stroke, or after the Stroke. Essentially, if not specifying the presence of an event implies its absence, then the querying system can only return exact matches. In this case, it can only return records that consist of a Drug A prescription, followed by a Stroke.

Suffice to say, this is not what users typically report as their intention, nor is an exact match query system a particularly useful tool for searching real world datasets. Because of this, it is critical that the absence of an event be explicitly specified. The query interface must make this readily apparent either when users are specifying their query, or reviewing their results, or ideally, both.

4.1.4 Difficulty 4: Accessing "Does Not Occur" Relationships

Once users understood the need to explicitly specify an absence, they typically had very little difficulty integrating simple absence scenarios into meaningful queries. However, they frequently overlooked the fact that specifying the absence of individual events does not encompass the full range of possible "does not occur" queries. Consider a dataset of hospital transfers where the typical pattern of events is that the patient is admitted into the emergency room, then transferred to a specialty ward like neurology, then discharged (Figure 4.8).

Figure 4.8: Typical patient transfer sequence.

A researcher might be interested in patients who deviated from this pattern. Users would frequently attempt to answer this type of question using multiple queries involving different permutations of events or the absence of one or more of the individual events, such as in Figures 4.9 and 4.10.

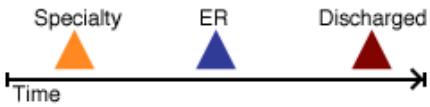


Figure 4.9: Different permutation of typical sequence.

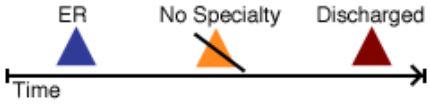


Figure 4.10: Event absent from typical sequence.

However, what they are really interested in is patient records in which the original sequence *does not occur*. Thus, the simpler approach is to search *for* the original sequence, and then select all the records that do not match that search. I found that users have difficulty transitioning to the mindset of using non-matches to their advantage. The query interface can play an integral role in whether this thought process shift is successful.

4.1.5 Difficulty 5: Understanding the Logic of Absences

From the onset, specifying the absence of an event cannot help but feel counter-intuitive. This is because the absence of a point event functions much like an interval, while the absence of an interval event can function much like a point. Consider first, the absence of a point event. It might be important for a researcher to find all patients who did *not* have a Stroke after taking Drug A. A Stroke is a

point event: it occurs at a single point in time. However, it does not make sense to search for the absence of a Stroke at any single point in time. There will, of course, be many points in time when the patient is not having a Stroke. The implication of this query is that a Stroke does not occur during the *entire* interval of time following the Drug A prescription. That is, the absence of a Stroke implicitly starts when the patient stops taking Drug A, and extends to the end of the patient record (Figure 4.11). This is referred to as a "spanning" absence, as it spans out to the presence events on either side of it.

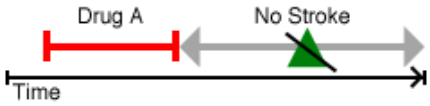


Figure 4.11: The implied end-points of a point event absence.

Counterintuitive? Yes, but users correctly report that this is how they expect the absence of a point event to behave, because this is the only way that the absence of a point event *can* behave. A point event absence is a duration of time, implicitly framed by the events specified on either side of it. Regardless of this, most users expressed this absence without a duration (Figure 4.12).

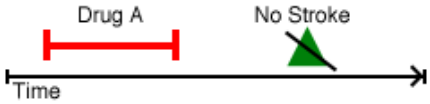


Figure 4.12: User specification of point event absence.

The problem with this specification is that, unlike the absence of a point event, the absence of an interval *can* be specified at a single point in time. For example, a researcher might be interested in patients who stopped taking their medication at any point after having a stroke. In this case, there need only be a single point in time when the interval is not taking place, such as in Figure 4.13.

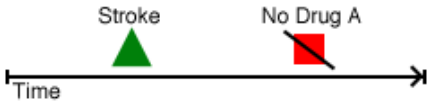


Figure 4.13: The absence of an interval, occurring at a single point.

The absence of an interval can also occur over a duration of time with implicit end points, just like a point event absence. For example, a researcher might be looking for patients who had a Stroke and then never took Drug A (Figure 4.14).

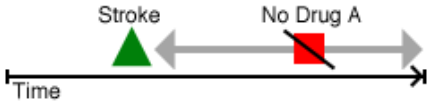


Figure 4.14: Interval absences can also have implied end-points.

Finally, the absence of an interval may only be relevant in the context of another event. For example, a researcher might be interested in patients who had a stroke when they were not taking Drug A (Figure 4.15).

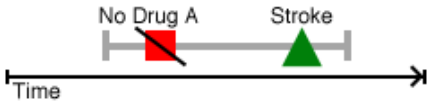


Figure 4.15: The absence of an interval occurs only during a specific event.

Thus, in order to distinguish the type of interval absence that is being specified, the interface must clearly represent the duration, or lack thereof, of all absence specifications. It is not sufficient to represent the absence of a point event as simply a slashed icon, as this representation is used for the absence of an interval that occurs at a single point. Users frequently reported that they had not considered, or were unaware of these different incarnations of an absence. The interface then, is responsible for helping them generate a query that conveys the absence that they intend to capture. No previous work in either temporal logic or temporal querying has explicitly addressed this logical inconsistency that arises with absences, or the effect that it has on query construction.

4.1.6 Design Study

To determine how a graphical query language might help to eliminate these difficulties, I conducted a generative design study - the first phase of a two-part user study - in which 20 computer science graduate students were asked to draw out

(with colored markers on paper) a series of temporal queries. The participants, all of whom self-rated as expert computer users, but with limited experience with databases, command-based query formulation, and temporal event data, were first introduced to the basic event representation in EventFlow (see the Timeline panel in Figure 4.17). They were then given a blank timeline and asked to draw out the following event sequences:

1. The patient is Admitted, then begins taking Drug A. The patient has a Stroke while taking Drug A.
2. The patient takes Drug A for at least 3 days.
3. The patient has a Stroke, then starts taking Drugs A within 5-10 days after having the Stroke.
4. The patient has a Stroke, then is Admitted.
5. The patient has a Stroke, then is Admitted, WITHOUT being Diagnosed in between.
6. The patient is Admitted, then has a Stroke. The patient is NOT taking Drug A when the stroke occurs.
7. The patient is not taking Drug A at some point before having a Stroke.

These queries were explicitly devised to target the problem areas that were encountered during the initial tests and interviews, in order to determine how users graphically distinguish between scenarios when they are made explicitly distinct. For example, it is impossible to express the absence of a Diagnosis in Query #5 as a non-presence without the query being identical to Query #4. The freehand graphics generated by the participants, were used to inform the visual query language used in the advanced query interface (discussed in detail in the Advanced Search section). Participants were also asked to explain their design choices during a subsequent debriefing. The details of this study are included in Appendix B.

4.2 Basic Search (Menu-Based)

The development process began with an interface for basic temporal search. The primary goal of this interface was to give users quick and easy access to three of the most fundamental temporal relationships: before, after, and during. The basic search is also designed to serve as an introduction to more complex temporal relationships. Use of this interface allows users to explore both interval events and absences without the complexity of additional temporal constraints. As such, this interface was integrated into the main control panel of the EventFlow application. It is described here, in the context of Hearst's search model:

4.2.1 Specification

The basic search interface consists of two separate modules: the subsequence module and the overlap module. Both modules consist of a simple list of drop-down menus which allow users to specify a series of event types (Figure 4.17). In each menu, users can select either the presence (denoted by a either a triangle or square icon for point and interval events respectively) or the absence (denoted by a slashed icon) of any event type. For interval events, the menus only allow users to specify complete intervals. That is, users cannot specify only the start or the end of an interval. This prevents users from trying to access "during" relationships using a sequential strategy (Difficulty #2).

Additionally, since the absence option appears next to its corresponding presence option in each menu, users are introduced to the concept of explicitly specifying absences by simply opening the menus (Difficulty #3). To remain consistent across both point and interval absences, all absences are handled uniformly, using implicit end points, since that is the default for point absences (Difficulty #4). The other interval absence options can be accessed in both the overlap query module and the advanced query interface.

```
select distinct t1.patid,
       t1.drug,
       t1.dispense_date,
       t1.next_drug,
       t1.next_dispense_date
from (select distinct patid,
       dispense_date,
       lead(dispense_date,1) over (order by patid, dispense_date, drug
          ) next_dispense_date,
       drug,
       lead(drug,1) over (order by patid, dispense_date, drug)
          next_drug
from  DRUG_TBL
where drug in('DRUG A', 'DRUG B')) t1,
       EVENT t2
where t1.patid = t2.patid and
       t2.ICD9 = 'STROKE'      and
  ((t1.drug = 'DRUG A' and t1.next_drug = 'DRUG B') and
          (t1.dispense_date = t1.next_dispense_date) and
          (t1.next_dispense_date < t2.event_start or  t1.
             dispense_date > t2.event_end));
```

Figure 4.16: SQL specification for patients that did not have a Stroke while taking Drug A and Drug B (the same query being executed on the right side of Figure 4.17).

The subsequence module targets before and after relationships in temporal event data. That is, it searches for disjoint, sequential events. For example, a researcher might be looking for records in which the patient took Drug A, followed by Drug B, and did not have a Stroke between these two drugs. This query, specified using the subsequence module is shown on the left side of Figure 4.17.

By contrast, the overlap module targets "during" relationships, or events that are happening concurrently. The goal of this module is to provide a shortcut to the nine interval relationships that are encompassed by this very common, plain English description (Difficulty #1). The right side of Figure 4.17 depicts an overlap

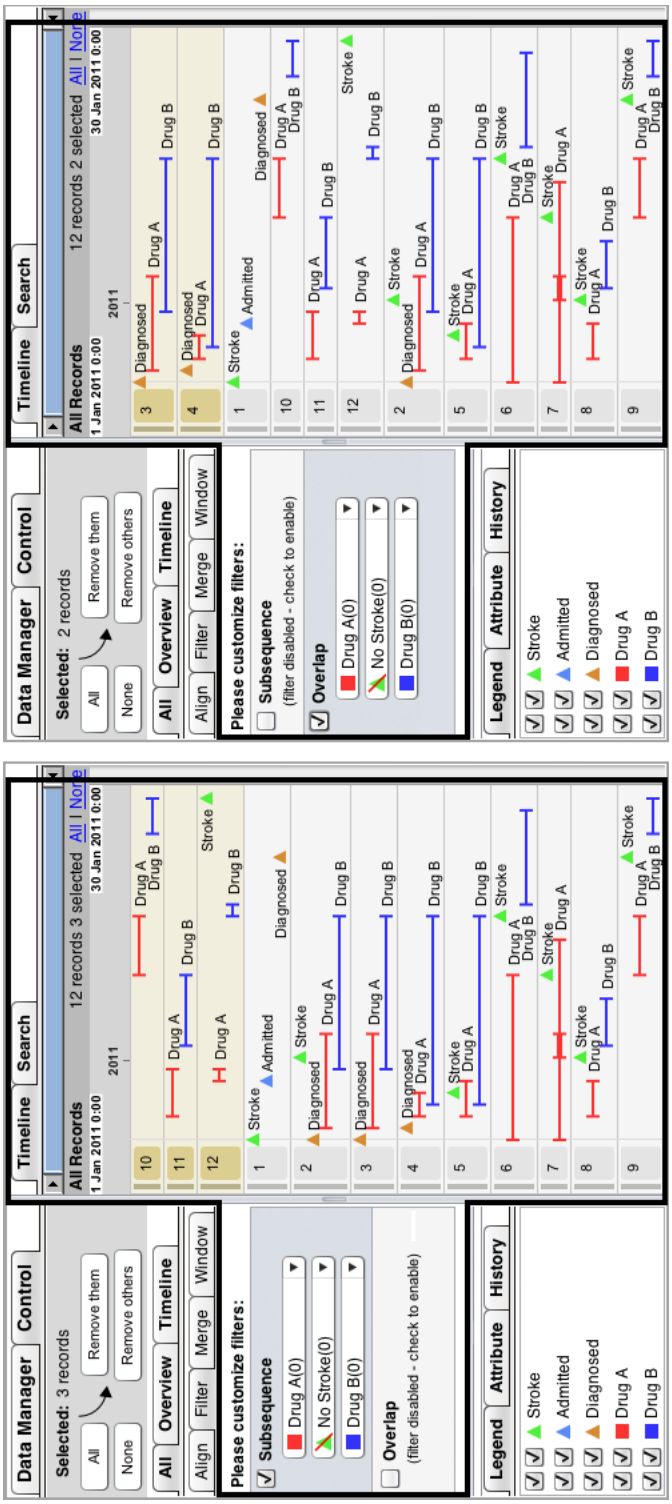


Figure 4.17: Basic Search: The Subsequence module (left) targets before and after relationships. The Overlap module (right) targets during relationships. Query results, with matching records highlighted at the top of the list, are displayed in the Timeline panel on the righthand side of both figures.

search for records in which the patient does not have a Stroke while taking both Drug A and Drug B. As opposed to the subsequence module, which can match events spanning across a patient's entire record, the overlap module searches only for a single point in that record when the specified events are occurring concurrently. Again, by having two separate modules with specialized functionality, users must mentally shift their search strategies, based on the event relationships they are accessing.

To illustrate the complexity of this seemingly simple overlap query, my partners at Oracle constructed the same query using SQL (Figure 4.16). The overlap query module distills this entire block of code into three simple menu specifications.

4.2.2 Presentation of Results

As mentioned previously, the basic search interface is integrated into the EventFlow visualization tool, which consists of two different panels for visualizing a set of temporal event records. One panel depicts a listing of each individual record in the dataset, laid out across its own timeline. A second panel depicts an aggregated view of the entire dataset, where records with the same event sequences are grouped together in a summarizing display. A more detailed description of the EventFlow display can be found in 6. For the purposes of this section, only the individual record panel is depicted.

When queries are specified using either the subsequence or overlap query module, matching records are selected and moved to the top of the individual display panel (Figure 4.17). This allows users to quickly see not only the records that were matched to their query, but also the records that did not match. A count of the number of selected records is displayed at the top of the control panel. While there is no explicit direction towards leveraging non-matches into effective queries, this strategy of keeping both the matches and non-matches visible at least prevents them from being forgotten altogether. If users identify the non-matches as the primary point of interest, they can invert the selected records (thus selecting all

the non-matches) in a single click (Difficulty #4).

Additionally, the overlap module gives users the option to highlight areas of the aggregated display that have matched the overlap search. This provides a sense of the frequency and duration of these overlaps without requiring use of the advanced search. Users can select the highlight color of their choice, and can strengthen the effect by reducing the overall interval opacity. A demonstration of the highlighting feature can be found in Figure 4.18.

4.2.3 Reformulation

In the basic search interface, queries are reformulated by simply reconfiguring the selected items in the drop-down menus of the appropriate module. The configuration from the initial query remains in place unless the module is explicitly reset. Users can quickly return to their original query in order to make the necessary adjustments. Users also have the option of removing either matches or non-matches from the display before they specify a new search. Thus, their view of the dataset can be narrowed down as their exploration progresses.

4.3 Advanced Search (Graphic-Based)

While the basic search interface gives users easy access to either before and after relationships (Subsequence module) or during relationships (Overlap module), the advanced search allows them to specify these relationships in tandem, as well as access more complex temporal features such as absolute time constraints, the full range of absence scenarios, and flexible event matching. The advanced search interface revolves around a visual query language that is used to draw the desired sequence of event relationships. Wongsuphasawat et al. [124] found that users prefer this graphical method over the menu-based interface for specifying complex temporal relationships.

The development of the advanced search began with an interface closely mod-

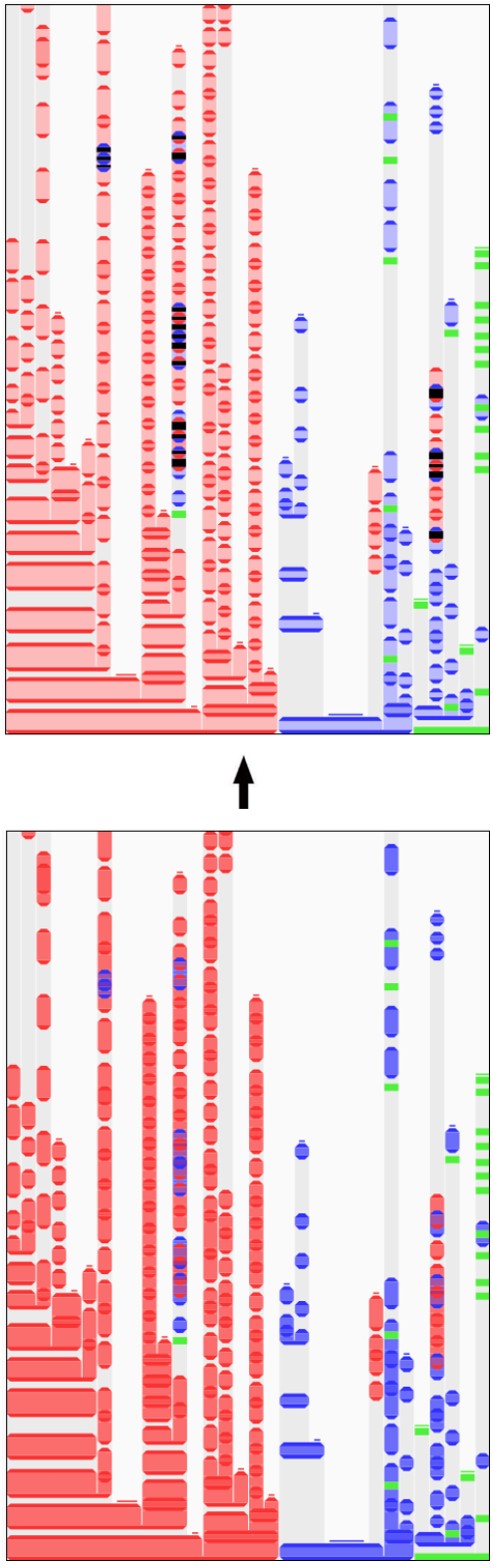


Figure 4.18: To quickly identify overlaps between the red and blue intervals in a complex dataset (left), black highlighting and reduced interval opacity can be specified (right).

eled after the search interface used in Similan [124], which was augmented to include an initial implementation of both absences and intervals. The interface was then updated incrementally with improvements gleaned from both the hand-drawn, generative phase of the user study and a second, usability phase:

In this phase, the 20 participants were asked to construct the same set of seven queries using the advanced query interface. I observed both the users' understanding of the graphics and their interactions with the interface, loosely following the Production-Preference-Performace program for developing graphical representations of abstract concepts [48]. The findings are discussed throughout the remainder of this section.

4.3.1 Specification

Queries are specified by adding query elements to a designated canvas. Users click on the canvas to bring up the Element Editor Panel (Figure 4.20), where they can specify the details of each query element that is added to the canvas. They can also select groups of elements on the canvas to specify additional constraints. The interface supports the following features and constraints, which can be seen in Figure 4.19, and is discussed in further detail in Section 4.3.4:

- The presence of point and interval events (1).
- The absence of point events (2).
- All three absence scenarios for interval events (3).
- Time constraints between events (4).
- Event attribute matching (5).
- The presence of wild card events, which allows a query element to be matched to an event of any category (6).

- The absence of wild card events, which dictates that no event can occur between two query elements (7).
- Flexible event ordering (8).
- Event and event pattern repetition (9).

The hand-drawing phase of the user study greatly informed the graphics used in the advanced query interface. For example, absences were initially displayed as individual gray boxes, interspersed between the presence events. However, multiple participants designated a separate section of the canvas for displaying absences, a technique that was very effective for keeping the display simple. This idea was quickly integrated into the evolving design. Additionally, interval events were initially displayed as double-sided arrows. However, many users felt that the pointed ends expressed the continuation of the interval, rather than its end-points. The interval graphic was thus changed to be a connected pair of bars. Similar to the concept behind the basic query interface, when users add intervals to their query, a fully-formed interval is added to the canvas. That is, users cannot add only an interval start-point or an interval end-point to their query (Difficulty #2).

Participants had no difficulty using the advanced query interface to specify sequences of presence events, including more complex temporal constraints such as interval durations and gaps between events. The initial implementation of absence specification, however, involved a series of options that allowed users to specify whether the absence should have implicit end-points, occur at a single point in time, or apply only to a specified presence event. Users had difficulty understanding these options, regardless of the language used in the description. To mitigate this issue, the absence specification form was narrowed down to a single checkbox which controls whether the duration of the absence automatically spans out to the nearest events on either side. For interval absences, users can uncheck this box in order to have manual control of where the absence's end-points are placed (Difficulty #5).

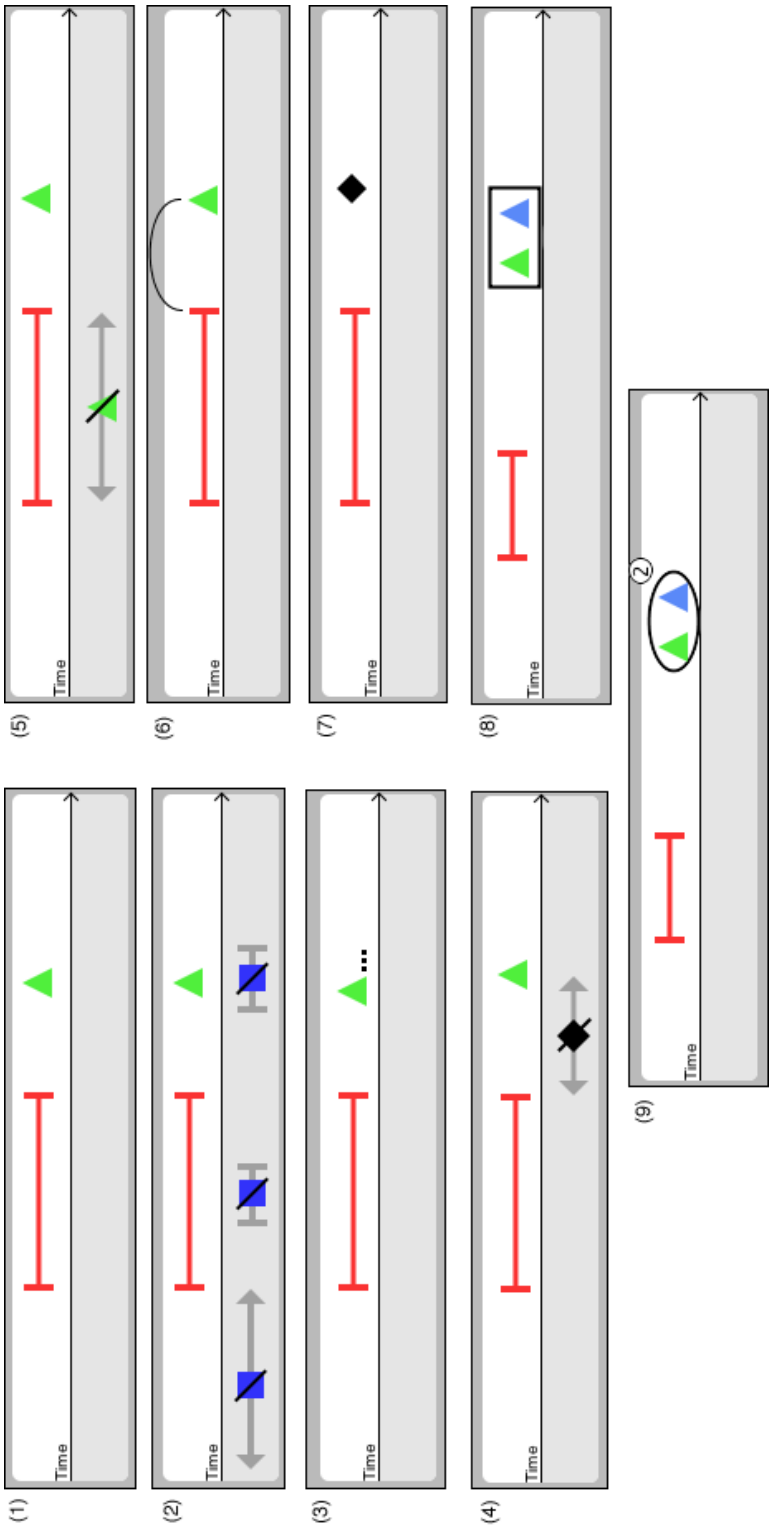


Figure 4.19: (1) The presence of point and interval events. (2) The absence of point events. (3) Event attribute matching. (4) The absence of wild card events. (5) The three absence scenarios for interval events. (6) Time constraints between events. (7) The presence of wild card events. (8) Flexible event ordering. (9) Event and event pattern repetition.

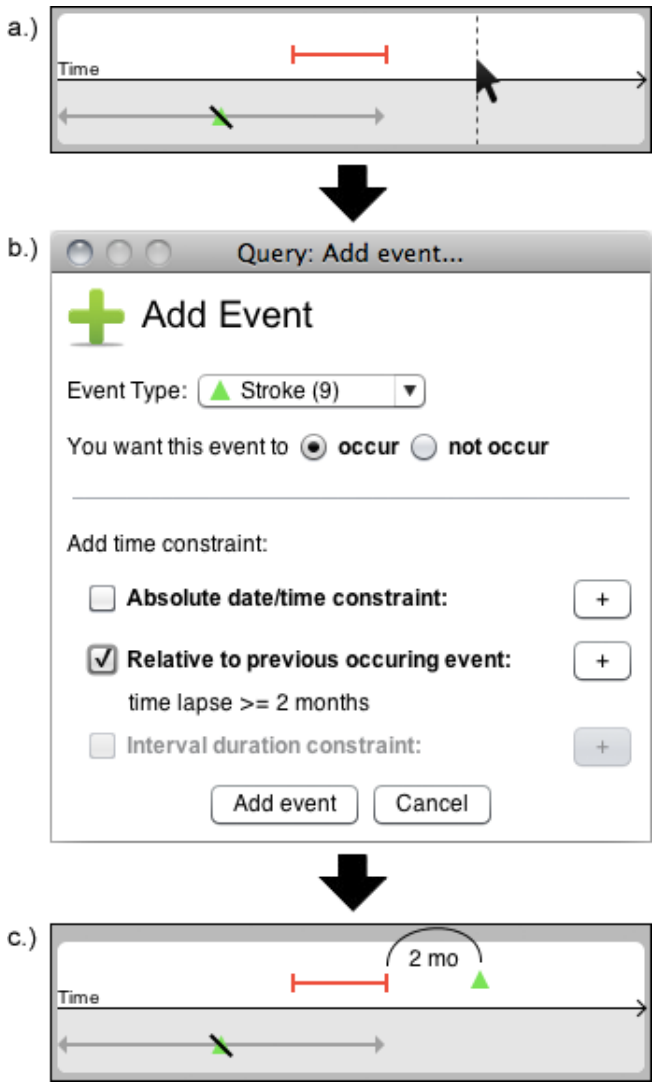


Figure 4.20: Specifying a query: Users add query elements by clicking the desired spot on the canvas (a), and adjusting its features using a pop-up form (b). The element is then added to the canvas, where it can be dragged, deleted, or modified (c). Event details can be seen on hover.

4.3.2 Presentation of Results

Similar to the basic search result presentation, matching records are moved to the top of the record display. However, the advanced search explicitly separates matches and non-matches into two separate lists, and ranks records in each list by how closely they match the query (Difficulty #4). Matches are ranked by the

number of extra events in the record (events that were not matched to an element in the query). Non-matches are ranked by how many query elements were matched within the record. A full display of the advanced query interface can be found in Figures 4.25 and 4.26.

These separate, ranked lists allow users to quickly find the most likely false positives (which will be towards the bottom of the matched list) and false negatives (which will be towards the top of the non-matched list). None of the participants in the study had any difficulty evaluating whether or not their query had returned correct results, indicating that the result presentation conveys enough information to accurately identify errors.

4.3.3 Reformulation

The advanced search interface makes it extremely easy for users to modify their queries. Elements on the query canvas can be dragged into new positions or deleted. Users can also click on any element to bring up its Element Editor, or hover over the element to view the event details. The only difficulty that participants encountered in the Reformulation stage was that, if they missed the element they were trying to click on and clicked on the background instead, the form to add a new query element would appear. This confused users, who were expecting to see the form to modify an existing element. This problem was solved by making the icons slightly larger and adding a more visible hover effect.

An additional feature of query reformulation in the advanced search is the ability to additively or subtractively select the result sets. Both the matching and non-matching result sets give users the option to either select all of the results in that set or deselect all of the results in that set. The key is that this selection remains in place as users execute additional queries. This makes it easy for users to refine their results using multiple queries. For example, a user might select all the matching results from an initial query, and then run a second query and deselect the matching results. This will prune the initial selection down to only the records

that matched the first query, but did not match the second. An example of this feature being used in practice is describe in Section 4.4.

4.3.4 Advanced Query Scope

The EventFlow visualization is designed to support the analysis of event patterns and sequences, with on-demand access to additional details such as metric comparison and event attributes. Because of this, the scope of the advanced query system is structured similarly. The primary focus of the query system is to allow users to express any query that can be reduced to a finite sequence of inequalities using the operators "<" and "=." An example of such a reduction is shown in Figure 4.21.

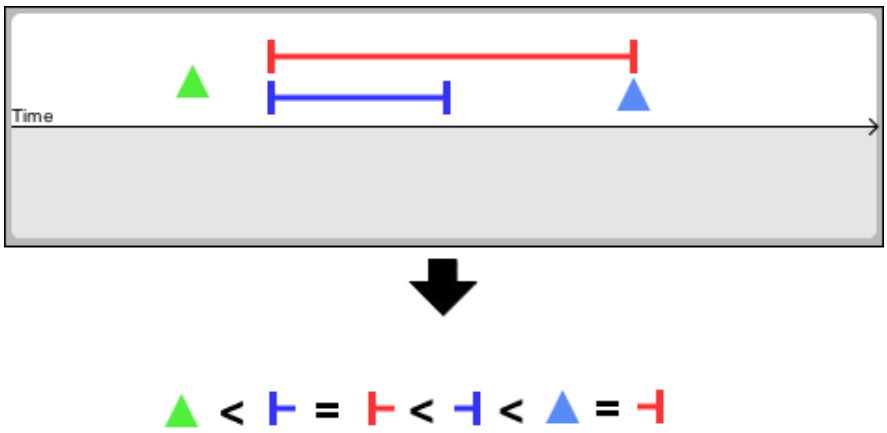


Figure 4.21: EventFlow's advanced query system supports any query that can be reduced to a finite sequence of inequalities.

Given this scope, the advanced query system supports the specification of all 13 of Allen's interval relationships, as well as the eight additional relationships that can be derived from considering point events as well. These 21 relationships are listed in Figure 4.22. In relation to EventFlow's visualization, the advanced query systems allows users specify any event sequence that would produce a visually distinct branch in the aggregated display.

The query scope is further extended to account for the absence of an event.

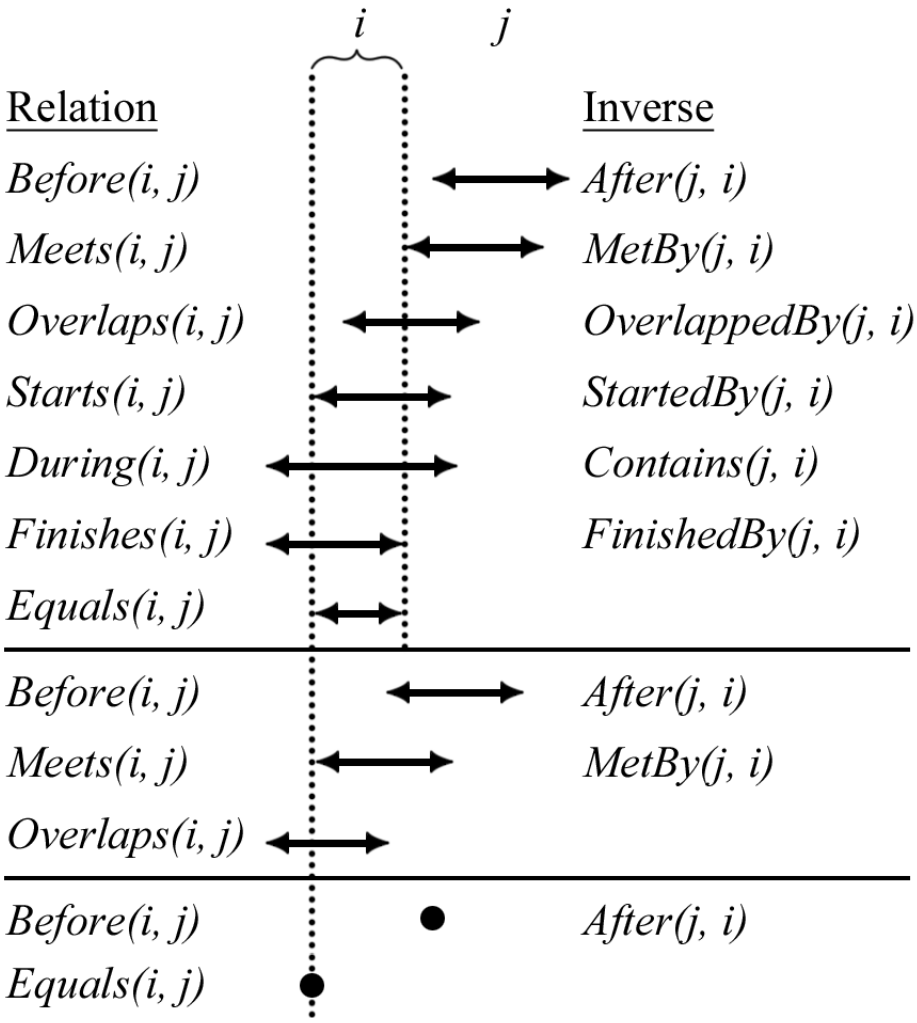


Figure 4.22: Allen's 13 interval relationships (top), augmented to include point-interval relationships (middle), and point-point relationships (bottom).

As discussed in Section 4.1.5, absences can either span the entire gap of time between two specified presence elements in the query or, in the case of interval events, occur at a single point in time. In this latter case, the absence can be logically translated into a spanning absence between two presence events, even if those events are not explicitly specified in the query (see Chapter 5, Section 5.3.7). This allows absences to be incorporated into the inequality sequences that are used to capture queries for the presence of events (see Figure 4.23). Overall,

this gives EventFlow's advanced query system comprehensive coverage over any event pattern involving points, intervals, and absences that can be expressed as a finite sequence of inequality relationships. It is the first graphic-based system to offer this depth of scope. From this baseline, which was chosen to reflect the focus of visualization, three additional considerations were added to the query scope: metric constraints, event attributes, and common shortcuts.

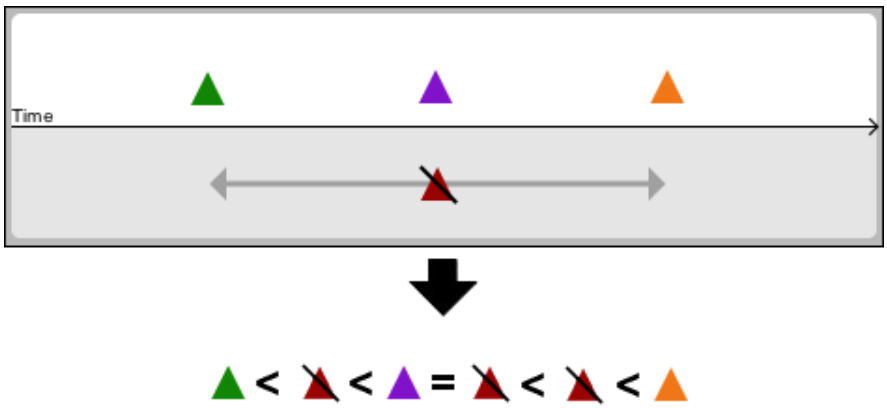


Figure 4.23: The absence of an event can be incorporated into the same sequence of inequalities.

4.3.4.1 Metric Constraints

While event patterns are given the foremost attention in EventFlow, the metric relationships between events remain a critical consideration [47]. In the visualization, users can access this information by hovering over the time lapse between any two events. Accordingly, users can also incorporate these metric relationships into their queries by augmenting any of the inequalities being expressed between the presence events in their query with upper and lower metric constraints. For example, a user might specify that Event B must occur at least 3 days after Event A, but within 8 days. However, users cannot currently specify metric constraints relating to an absolute timeline. For example, users cannot specify that Event B must occur at least 3 days after February 5th, 1985. Again, this limitation reflects

the scope of the visualization. If users want to incorporate exact dates into their queries, they must add these dates as explicit events in their datasets. This can be done in EventFlow using alignment and marker events.

Additionally, metric constraints relating to the absence of an event are not directly supported. For example, users cannot specify that the absence of a medication must occur for 3 days before a patient experiences a symptom. This is because absence is logically dependent upon presence. Any metric constraint relating to the absence of an event can be translated into a metric constraint involving the presence of an event. For example, the wildcard events discussed later in this section can be used to capture the absence of a medication that occurs for 3 days before a symptom (see Figure 4.24).

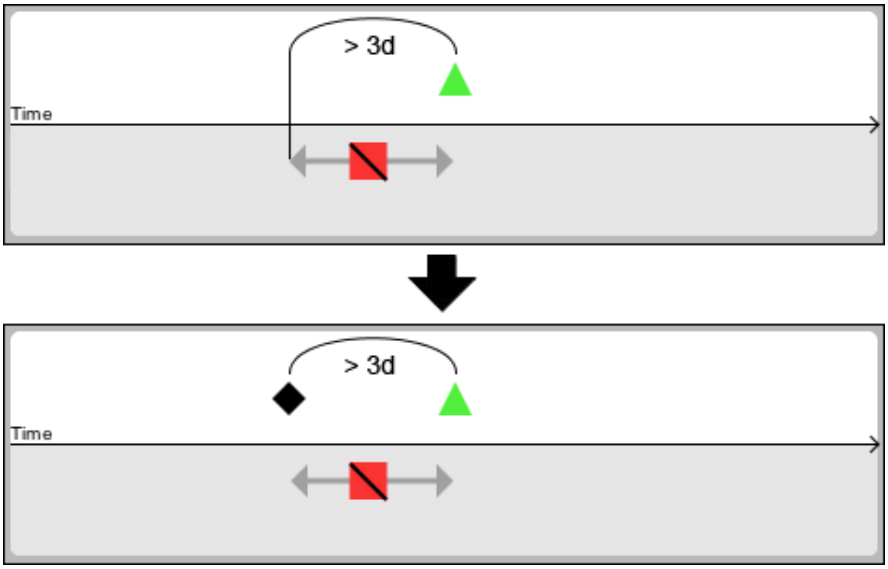


Figure 4.24: Any metric relationship involving the absence of an event can be translated to a metric relationship involving the presence of an event.

4.3.4.2 Event Attributes

Access to event attributes is currently not supported in the advanced query system. To a large extent, this limitation is due to the more pressing need for other features, and should be considered in the future development of the query system.

However, this feature was not a priority because, much like metric constraints over absences, users can access event attributes through the data transformation techniques described in Chapter 6. Attributes can be actualized into real events that can then be accessed through query.

4.3.4.3 Common Shortcuts

Users can currently specify any finite sequence of "<" and "=" relationsips using EventFlow's advanced query system. However, there are many common temporal relationships that are comprised of multiple event sequences. For example, there are nine relationships between two interval events in which the intervals are considered to "overlap." The advanced query system allows for the results from multiple queries to be combined into a single selection, so in theory, users could access "overlap" relationships by specifying all nine event configurations, and combining the results.

In practice however, this is a considerable burden and, in most cases, users are not able to generate the complete set of sequences that capture the intended temporal notion. Because of this, wildcard events, flexible event ordering, and event pattern repetition were implemented as shortcuts to commonly needed sets of event sequences. These features were borrowed from analogous features in string matching. They do not extend the query scope in any way, but allow users to execute multiple queries in a single specification. Unlike the analogous features in string matching, however, the flexible event ordering constraints and event pattern repetition constraints cannot be nested. EventFlow only allows users to utilize these shortcuts on disjoint sequences of events.

4.4 Case Study Examples

Once the development of the basic and advanced search interfaces was complete, I spent one-on-one time with two different epidemiologists at the PVC, using Event-

Flow and both temporal query interfaces to explore the evolving questions of their on-going research studies. While both of these studies evolved into longer term collaborations that are described in further detail in Chapters 6 and 7, my initial focus on EventFlow's querying systems is described here.

4.4.1 Opioid Use

The first epidemiologist was investigating patterns of opioid use. Opioids are typically prescribed to treat pain, and range in strength from low, over-the-counter doses, to high, potentially-habit-forming prescriptions. Additionally, these medications can be prescribed as short-term, "acute" treatments, or long-term, "chronic" treatments. The epidemiologist was interested in the relationships between the dosage strength and the treatment duration. For example, are low doses of opioids being prescribed for acute treatments?

Her dataset was organized into two layers. The first layer consisted of each patient's individual opioid prescriptions, categorized as either a "high," "medium," or "low" dose. In the second layer, individual prescriptions in close proximity to one another were combined into "eras," and categorized as either an acute or chronic treatment, based on the duration. Thus, each prescription was represented twice, once as an individual dosage, and once as a component of a treatment. Hospital visits were also included in the dataset as potential catalysts for treatment changes. Overall, the dataset consisted of 500 patients and 2171 events.

The patients that the epidemiologist was initially interested in were those who received a high opioid dosage as part of an acute treatment. To isolate these patients, she used the overlap search module to select patients with concurrent "high" and "acute" intervals. From there, she was interested in cases where this combination was not closely preceded by a hospital visit. It was not readily apparent to her how to construct this query (Difficulty #4). I guided her to construct a query for cases where this combination *was* closely preceded by a hospital visit. From this hint, she quickly realized that the records not matching this query would satisfy

her original query. She was able to isolate these records without further assistance (Figure 4.25).

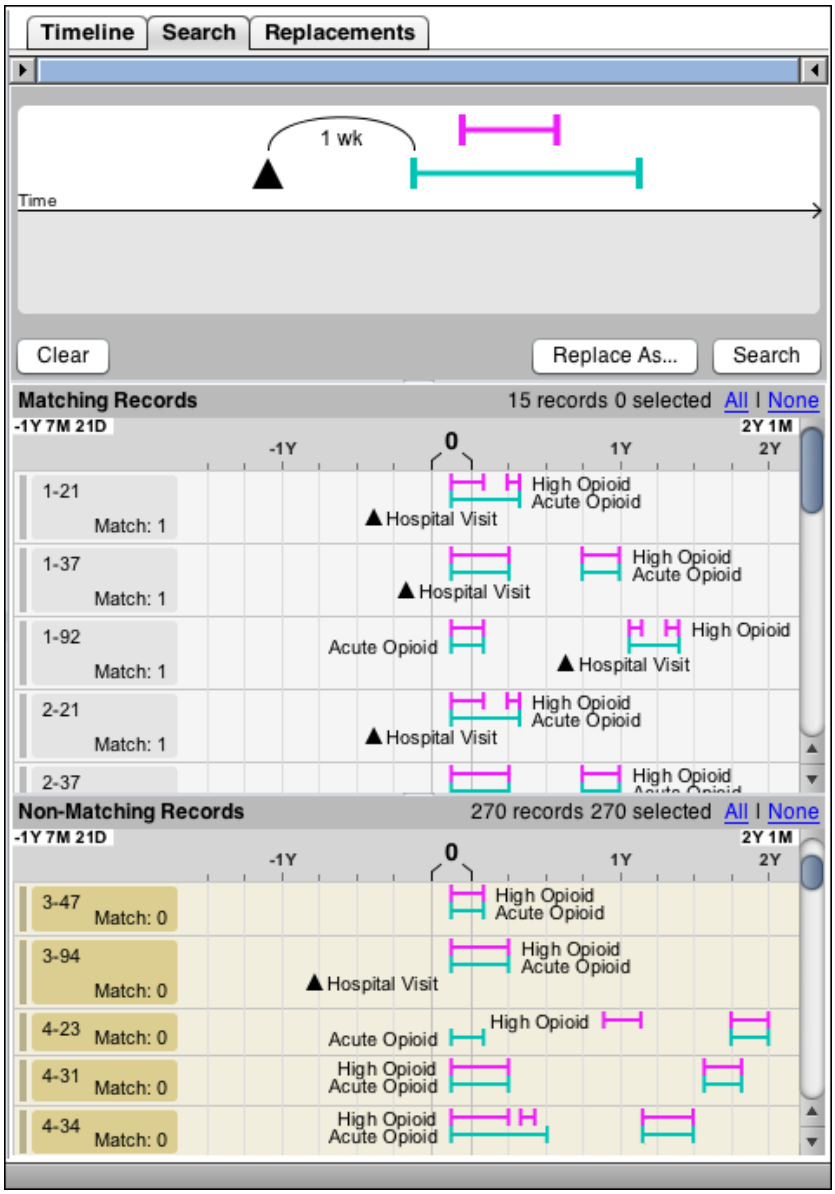


Figure 4.25: The Advanced Query Interface, including the query canvas (top), matching records (middle), and non-matching records (bottom). In certain cases, it is easier to access a result set by designing your query around the records that will *not* match. Here, the non-matching records are selected.

4.4.2 Asthma Medication Prescribing Practices

The second epidemiologist was working to understand the prescribing practices of asthma medications. Much like opioids, asthma medications range broadly in the strength of the prescription. One of the strongest classes of asthma medications, the Long-Acted Beta Agonist (LABA), should only be prescribed when other, weaker asthma medications have proven ineffective. This pattern is described as the "step up" to the LABA prescription. Furthermore, when a LABA is prescribed, it should be followed by multiple prescriptions of successively weaker asthma medications, known as the "step-down."

One of the queries that the epidemiologist constructed to explore these prescribing practices was to search for patients who were prescribed a weaker asthma medication within 3 months of both the start and end date of their LABA prescription. However, using the advance search, this query also returned patients who had been prescribed a LABA either before or after the query sequence (Difficulty #3). The epidemiologist immediately noticed this error in the results display, and modified the query to include the specification that there must not have been LABA prescriptions before the initial, weaker prescription, or after the final prescription. The final query, and the results it produced, can be seen in Figure 4.26.

This case study also illustrated the capability of EventFlow's aggregated display to prompt users to think about additional interval relationships (Difficulty #1). The aforementioned query only produced four matching results in a dataset of 100 records. The epidemiologist had gotten similar results when she had run the analogous query using command-based tools. While she felt that this number was low, she had previously had no reason to question it. However, when this query was run in EventFlow, the epidemiologist immediately noticed a second prescription pattern in the aggregated display that she had not previously considered. In these records, patients had been prescribed a weaker asthma medication that started prior to their LABA prescription, and extended beyond the end of the their LABA

prescription (see Figure 4.27).

The epidemiologist had not even been aware that this pattern existed in the dataset. She quickly formulated a second query capture this relationship, which resulted in an additional seven matching records. By maintaining the accessibility of non-matching query results, EventFlow allowed the epidemiologist to more than double her result set. She was also made aware of this second prescription pattern, which she integrated into all of her future queries against this dataset.

4.4.3 Summary

In both of these case studies, the epidemiologists remarked that EventFlow, and both the basic and the advanced query interface, were much easier to learn and use than the command-based, statistical software that they had been using previously. Both of the query interfaces prevented the epidemiologists from constructing queries in which intervals were treated as disjoint point events (Difficulty #1). However, further work may be needed to fully determine whether the interfaces are effective at distinguishing between different absence scenarios (Difficulty #4), as I have yet to encounter the full range of such scenarios within a single case study. At the prompting of these epidemiologists, the PVC is currently working to install EventFlow on the Pentagon servers to facilitate their use of the software.

4.5 Summary

This chapter introduced two novel interfaces for constructing and executing temporal queries involving intervals, absences, and a whole host of other constraints. I describe four of the most common difficulties that users experience when formulating queries involving intervals and absences, and described how the query interface can play a central role in alleviating these difficulties. EventFlow's query interfaces facilitate temporal search in four primary ways that set them apart from standard, command-based querying techniques:

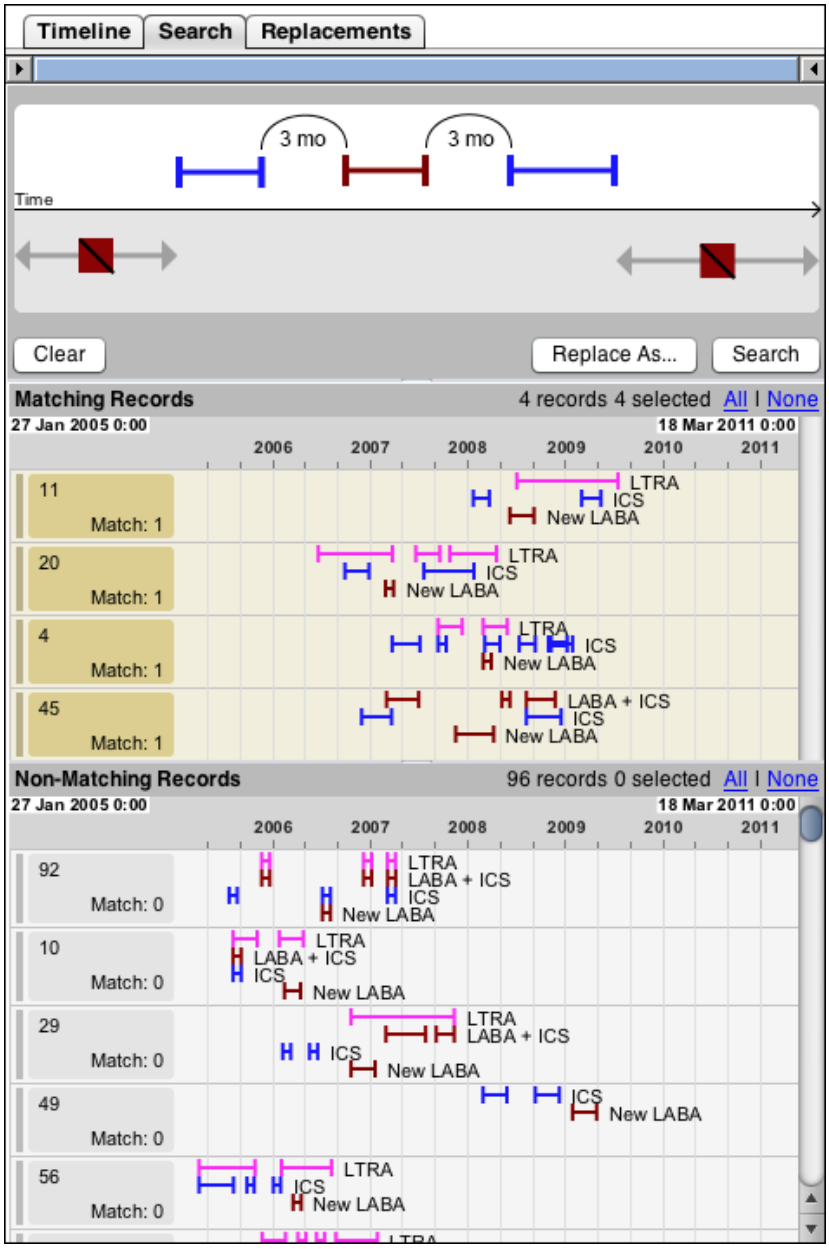


Figure 4.26: In this query, the researcher was looking for patients who received a weaker asthma medication (in blue) within 3 months of both the start and end date of a stronger asthma medication (in red). They also wanted to ensure that this sequence was neither preceded nor followed by the strong medication.

1. Easy to learn and use.

2. Offer access to a wide range of temporal relationships.

3. Guide users away from common difficulties.

4. Recover from inevitable errors.

5. Address all three stages of Hearst's search model.

The use of both interfaces was illustrated in two case studies with epidemiologists at the US Army Pharmacovigilance Center, who used the query interfaces, as well as the encapsulating EventFlow visualization tool, to answer questions involving prescribing practices and medication interactions. In these sessions, the epidemiologists were able to address their evolving research questions using both the basic and advanced query interfaces. They described the entire query process as a dramatically simpler alternative to the command-based approach to which they had been previously limited.

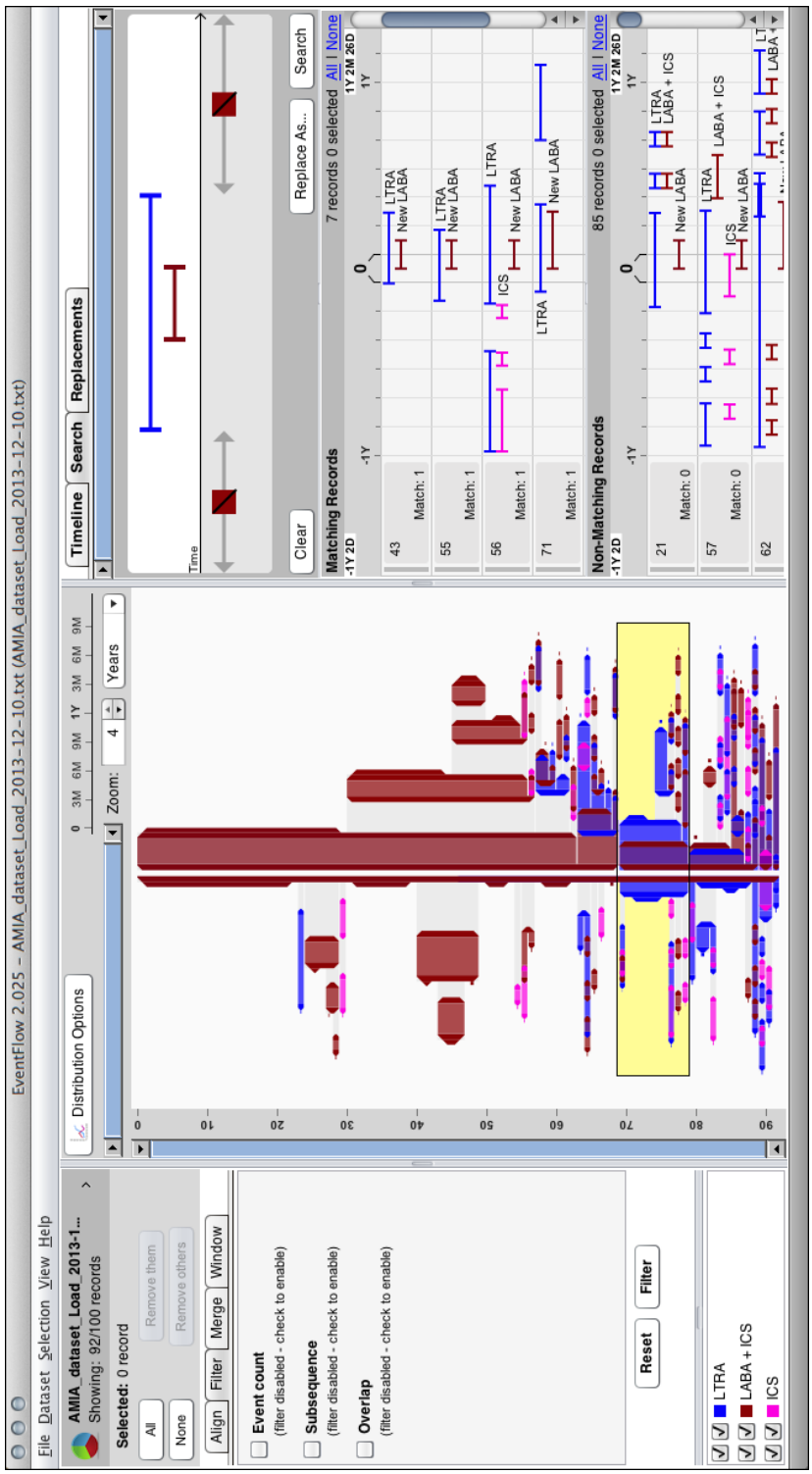


Figure 4.27: After removing the results from her initial query (Figure 4.26) from the display, the epidemiologist immediately noticed an additional pattern of interest (highlighted in yellow). She quickly formulated an additional query to capture these results (top left).

Chapter 5

Temporal Query Processing

The user-facing challenges of specifying temporal event queries can be so substantial that it is easy to lose sight of the fact that, from a computational perspective, these queries must still be processed on the back-end to deliver accurate results. By all accounts, this is a non-trivial task that, much like some of the difficulties described in the previous chapter, is masked by the simplicity of other back-end considerations. For example, the prevalence of temporal event data as a means of historical representation can be attributed, at least in part, to the simplicity with which it can be stored. While a detailed description of temporal event database storage can be found in Chapter 8, for all intents and purposes this data can essentially be stored in a four column, database relation:

Record ID	Event Category	Start Time	End Time

An alternate strategy is to separate the start and end times of interval events into different rows, but keep them linked by a unique identifier:

Record ID	Event ID	Event Category	Time

In either case, temporal event data is well-suited to the tabular structure of standard database relations, making it an ideal format for representing our daily life. The difficulty, however, arises when this data must be queried. Temporal event queries are notoriously ill-suited to standard, relational query processing. Answering such a query with standard relational operators results in a lengthy series of

Event Category	Record ID	Start Time	End Time
Drug A	Patient X	1/10/13	1/20/13
Stroke	Patient X	1/15/13	

Figure 5.1: The patient's Stroke event is recorded after the interval event, even though it occurs *while* the patient was taking Drug A.

Event Category	Record ID	Start Time	End Time
Headache	Patient X	1/15/13	
Stroke	Patient X	1/15/13	

Figure 5.2: Ordering is imposed even though the events are concurrent.

sorting and large self-join operations. This strategy does not scale well as datasets get increasingly larger. Furthermore, relational storage structures can obscure the order of events in the record, either by representing events out of order, or by imposing an order when none exists (see Figures 5.1 and 5.2). Because of this, extensive preprocessing can be required in order to properly access the event relationships.

There has been considerable work done to adapt standard relational query processing to better handle temporal event queries [54,92,128]. However, the extent to which this work is integrated into commercial systems and used in practice is not clear. The majority of the researchers with whom I've collaborated have been using standard relational database packages. In many cases, extensions to standard database querying focus on command-based shortcuts to access temporal relationships, and do not delve into how these queries are actually processed.

Ironically, temporal event queries are very similar to matching queries over strings, a data type that, unlike event data, is unanimously *not* well-suited to standard relational storage. String matching techniques are the inspiration behind many temporal data mining algorithms. They typically work by performing a linear scan of the event record, matching record events to the query elements as

they go (scan-and-backtrack) [51]. In general, these algorithms gain efficiency by minimizing the amount of backtracking that must be done when a potential match does not work out. The back-end of EventFlow's advanced query system was initially built with one such algorithm, developed by Wang et al. [117], which was extended to handle interval events and the additional backtracking that they incur. The overarching story of temporal event data then, is that it is perpetually in the awkward position of being conveniently stored in an environment that does not facilitate querying.

Then things get complicated. It is not always enough to query event records based on whether they do or do not contain a certain pattern of events. In many cases, there will be multiple ways to match the query pattern to the events in a record. This is particularly true of longitudinal medical records that contain multiple years of patient treatment history. In these cases, the query must not only determine whether there is a match, but must also determine the events that *best* match the query pattern (best-match event query). This is a pressing need in real-world scenarios such as patient similarity and process optimization. For example, a medical researcher might want to find the shortest hospital stay for a given patient, during which a certain medication was administered. Current data mining techniques focus primarily on the existential aspect of queries, but do not determine best-match.

In this chapter, I demonstrate that best-match event query is NP-Hard, and use this complexity to motivate an integer programming (IP) approach to solving these problems. Integer programs are an appealing solution for temporal event queries in general because they:

1. Require no ordering of the underlying event record.
2. Can easily discern between multiple possible solutions within a single record using the objective function.
3. Allow query patterns to be constructed using as many or as few ordering

constraints as necessary.

4. Can be constructed incrementally as queries are specified and reformulated.
5. Provides a natural metric for ranking results, again using the objective function.

This chapter is organized as follows: I begin with a discussion of how and why an IP solution is an appealing back-end for EventFlow's advanced query interface in Section 5.1. The hardness of best-match event query is discussed in Section 5.2. A full description of how a temporal event query is formulated into an integer program is presented in 5.3, followed by a simple example of such a formulation in Section 5.4. Finally, in Section 5.6, I explore the feasibility of this approach in practice using three example queries.

5.1 Query Specification

The integer programming strategies discussed in this chapter are designed to serve as the back-end processing for a EventFlow's advanced query interface. This interface, described in detail in the Chapter 4, allows users to draw out event patterns of interest. The goal of the IP formulation presented here was to match the functionality of this query specification interface. A listing of this functionality for reference, as well as the corresponding graphics that will be used throughout this chapter, can be found in Table 5.1.

EventFlow is an ideal testing ground for an IP back-end due to the scalability constraints of the visualization component of the software. In order to produce a readable visualization, EventFlow operates on datasets that consist of potentially tens of thousands of records, but with each record consisting of between only 20-100 events. For larger datasets, EventFlow is loaded with a sample subset, which is typically sufficient for users to identify and analyze the predominant trends.

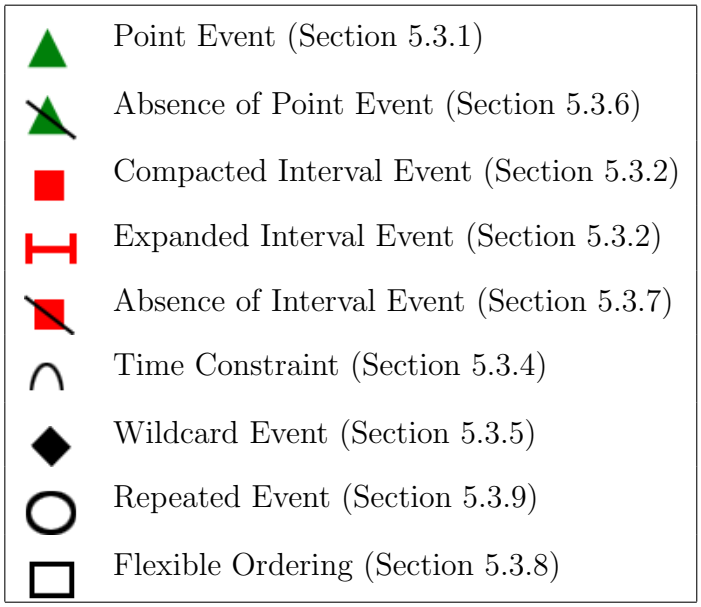


Table 5.1: Query graphics, for reference throughout this paper. Different colors indicate different categories of events.

Thus, the query processing strategy does not need to scale significantly beyond this "sweet spot."

From a usability standpoint, a critical feature of search is the ability to rank results. Ranking provides structure for the result presentation, as well as helps users identify possible false positives due to errors in their query. In EventFlow, the initial metric for ranking results is the count of extraneous events in the record that were not matched to the query. In many scenarios, however, this is not an informative metric. In fact, the ideal ranking metric is in continuous flux. Users execute queries, evaluate the results, and generate new questions, which in turn lead to new queries or perhaps even a different perspective on the same query. The appropriate metric for ranking query results can change as quickly as these questions evolve.

It is appropriate then, to choose a query processing strategy that naturally produces a ranking metric that is both flexible and informative. An IP back-end accomplishes this using the objective function. The objective function can not only distinguish between multiple possible matches within an event record, but

also distinguish the best match across records. The final value of the objective function can be used to rank the matched search results, and can be tailored to the users' current focus of analysis.

Additionally, EventFlow allows users to construct queries incrementally, by adding each element individually to the query canvas. Elements can be both modified and removed, and new queries can evolve from making slight adjustments to the previous query. This differs from many command-based query systems, in which a change to a single query element can alter the structure of the entire query.

This incremental query specification interface translates nicely into an integer programming back-end, where constraints can be added, modified, and removed in the same, incremental fashion. The constraints are almost entirely independent of each other (absences, which consist of paired constraints, are the one exception), and thus changes to one constraint will not require changes to the entire set of constraints. For example, if a user modifies their query to require that there is at least a 3-day time lapse between two pre-existing events, the resulting back-end change would modify only the time constraint between those two events. All of the other constraints would remain as they were.

5.2 The Hardness of Best-Match Event Query

The difficulty of selecting a back-end processing strategy for temporal event queries, and specifically best-match event queries, is that the hardness of the problem is highly dependent on the query itself. Consider, for example, the query depicted in Figure 5.3. The query is for two consecutive point events, and it can be assumed that the *best* match is the one that minimizes the time lapse between these two events.

Clearly this query can be processed against a given record by performing a linear scan across all of the events in that record. However, this strategy becomes less straightforward as the query gets increasingly complex. As query elements such

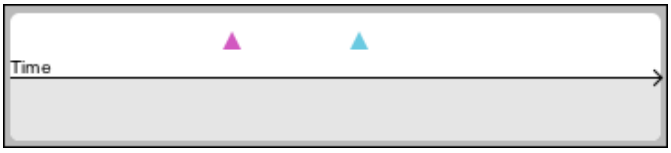


Figure 5.3: Query for two consecutive point events.

as absences and repetition are specified, linear scan strategies must be modified to allow for backtracking when matches for the first elements in the query are invalidated by subsequent event patterns [117]. These queries can extend the back-end workload to quadratic and cubic complexity.

When overlapping interval events are integrated into both the event record and the query, the hardness of best-match event query escalates even further. A series of overlapping interval events can be represented as an interval graph, where each vertex represents an event, and an edge is present between events that overlap each other (see Figure 5.5). Consider, for example, the query for three overlapping intervals, where the sequence of interval start and end points can occur in any order (Figure 5.4). When a query of overlapping intervals such as this is processed against an event record, the problem is equivalent to determining whether the interval graph of the query is a subgraph of the interval graph of the record (interval subgraph isomorphism).

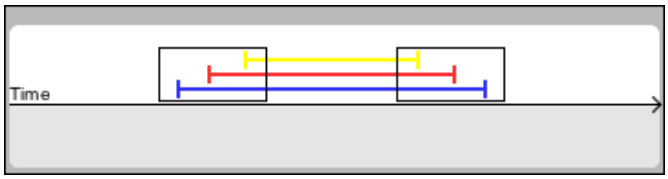


Figure 5.4: A query for three overlapping intervals, where the start and end points can occur in any order.

Much like temporal event queries, there are many varieties of the interval subgraph isomorphism problem, with equally varied hardness. Marx and Schlotter proved that induced subgraph isomorphism is NP-complete when graphs G and H are restricted to be interval graphs [23]. Heggernes et al. proved that when G is an interval graph and H is a connected proper interval graph, the problem

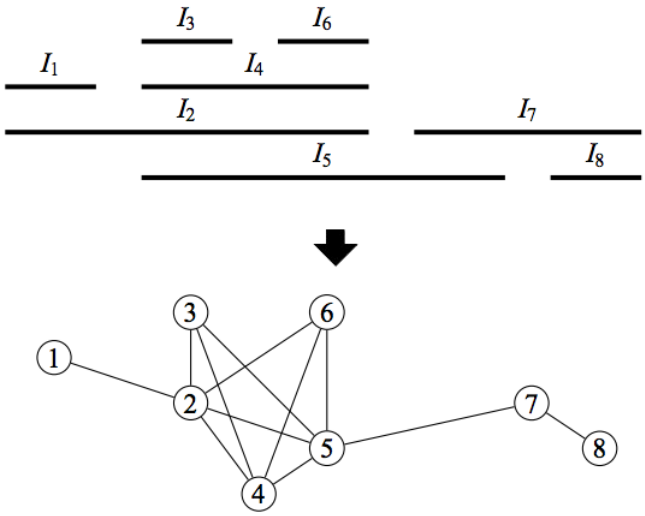


Figure 5.5: A pattern of overlapping intervals across a linear timeline can be translated into a graph representation.

is solvable in polynomial time [39]. Kijima et al. demonstrated that spanning subgraph isomorphism is NP-Hard for proper interval graphs using a reduction of 3-partition [49]. A simple, temporal pattern matching query without absences or other complex constructs (but with overlapping intervals) is equivalent to this last problem, and hence is NP-Hard.

Thus, a back-end processing strategy for temporal event queries must address a problem that can range from linear complexity to NP-Hardness. While many strategies strive to find a middle ground along this continuum, this chapter takes a top down approach by implementing an integer programming back-end, which is known to be NP-Hard. This strategy is applied to the more complex event queries, and then evaluated on its ability to scale down to simpler queries.

5.3 An Integer Programming Approach

In this section, the goal is to find the events in a given record that *best match* given a temporal event query. I demonstrate how to incrementally construct an integer program to solve temporal event queries. I begin with the most basic queries, those

that involve only a sequence of point events, and then progress through increasing complex constraints involving interval events, absences, order independence, and repetition.

5.3.1 Point Events Only

Consider a single record, consisting of events $E = \{e_1, e_2, e_3, ...e_i\}$, where each event is represented by two dimensions: time and category, i.e. $e_x = \{e_{x\_t}, e_{x\_c}\}$. Similarly, in this initial case, a query pattern consists of elements $Q = \{q_1, q_2, q_3.....q_j\}$, where $q_y = \{q_{y\_c}\}$, i.e. for each query element, an event category is provided. Since the query is specified across a single timeline, there is the additional constraint that $q_{y\_t} \leq q_{(y+1)\_t} \forall 1 \leq y < j$.

The query is processed by matching each element in Q to an event in E. This is done using an $i \times j$ table of binary variables $M = \{m_{1,1}, m_{1,2}, ...m_{i,j}\}$, where $m_{x,y} = 1$ if event e_x is matched to element q_y. Otherwise $m_{x,y} = 0$. This table will be referred to as the *match table*.

	q_1	q_2	q_3	$\cdots$	q_j
e_1	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$	$\cdots$	$m_{1,j}$
e_2	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$\cdots$	$m_{2,j}$
e_3	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$\cdots$	$m_{3,j}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
e_i	$m_{i,1}$	$m_{i,2}$	$m_{i,3}$	$\cdots$	$m_{i,j}$

The match table is subject to three primary constraints:

$$\{\forall 1 \leq y < j\}, \sum_{x=1}^{i} m_{x,y} = 1 \tag{5.1}$$

$$\{\forall 1 \leq y < j\}, \sum_{x=1}^{i} m_{x,y} \cdot e_{x\_c} = q_{y\_c} \tag{5.2}$$

$$\{\forall 1 \leq y < j\}, \sum_{x=1}^{i} m_{x,y} \cdot e_{x\_\mathrm{t}} < \sum_{x=1}^{i} m_{x,y+1} \cdot e_{x\_\mathrm{t}} \tag{5.3}$$

The first constraint, specified for every column of the table, is that each query element must be matched to one, and only one, event (Equation 5.1: **One-to-one Constraints**). The second constraint requires that a query element must be matched to an event of the same category (Equation 5.2: **Category Constraints**). Finally, the events must be matched in the correct temporal order (Equation 5.3: **Ordering Constraints**). For events that must occur at the same time, the "<" sign is replaced with an "=" sign. The timestamp, e_{x_t}, of the event that is matched to a given query element q_y will be referred to as q_{y_t}.

$$\sum_{x=1}^{i} m_{x,y} \cdot e_{x\_\mathrm{t}} = q_{y\_\mathrm{t}} \tag{5.4}$$

What is critical to point out is that, while some notion of order must be maintained across the query elements, there are no ordering requirements on the event record. A given record can be extracted from a database, and integrated into the IP as is. It does not need to be sorted or stored in any sort of custom data structure. Temporal relationships like *during* or *concurrently* are handled naturally by the IP constraints.

Additionally, in most cases, the number of constraints scales according to the number of elements in the query, rather than the number of events in the record. This is ideal because queries are typically shorter than the event records. While a patient might have 50, or even 100 events in their record, a query is not likely to contain more than 10 elements. Because of this, the number of constraints that are needed to formulate the IP will remain bounded.

5.3.2 Points and Intervals

With the inclusion of interval events, there is the additional constraint that if an interval's start point is matched to a query element, its end point must be matched to the corresponding end-point in the query. This can be accomplished by having duplicate entries in the match table.

Consider a small example in which (q_2, q_4) represents an interval's start and end point in the query sequence, and (e_1, e_3) represents an interval's start and end point in the event record. In this scheme, q_2 can be matched to e_1 if and only if q_4 can be matched to e_3. The previous table scheme results in the following constraint:

$$m_{1,2} = m_{3,4} \tag{5.5}$$

However, this constraint can be eliminated by simply allowing $m_{1,2}$ to appear twice within the table (note that these duplicates will never appear in the same row or column):

	q_1	q_2	q_3	q_4
e_1	$m_{1,1}$	$[m_{1,2}]$	$m_{1,3}$	$m_{1,4}$
e_2	$m_{2,1}$	$m_{2,2}$	$m_{2,3}$	$m_{2,4}$
e_3	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$	$[m_{1,2}]$
e_4	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$	$m_{4,4}$

These duplicate entries in the match table occur for every pair of query/event record intervals.

**Note: For the sake of simplicity, the duplicate entries in the match table should be assumed, but will not be explicitly stated in subsequent equations.

5.3.3 Event Attributes

A timestamp and a category (e_{x_t} and e_{x_c}) are the fundamental features of each event. However, events can have additional attributes, denoted e_{x_a}, that can represent anything from a prescription dosage to a temperature. These attributes can be easily incorporated into constraints of the IP. For example, the following constraint specifies that the dosage should be less than 30mg.

$$\sum_{x=1}^{i} m_{x,3} \cdot e_{x\_\mathrm{a:\ dosage}} < 30mg \tag{5.6}$$

5.3.4 Time Constraints

There are three primary time constraints that can be incorporated into a query: gaps between events, interval durations, and absolute dates. All of these can be incorporated into the IP using a modified version of the ordering constraints (Equation 5.3). For example, to specify that an event must occur at least 3 months after the preceding event, the ordering constraint would be augmented to:

$$q_{b\_\mathrm{t}} - q_{a\_\mathrm{t}} \geq 3 \text{ months} \tag{5.7}$$

5.3.5 Wild Card Elements

Similar to string matching, a wild card query element can be matched to an event of any category. For example, in a dataset of hospital transfers, a query could be formulated for patients who arrived to the Emergency Room, and then were transferred through at least one other department before being discharged (see Figure 5.6). Wild card events actually simplify the integer program. They require one-to-one and ordering constraints to be applied, similar to a normal point event, but do not require any category constraints.

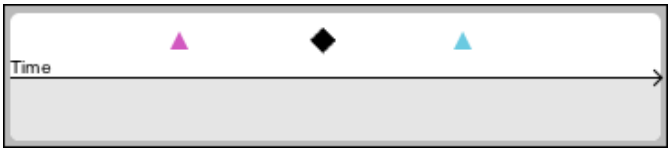


Figure 5.6: Query for patients who arrived to the Emergency Room (in pink), and then were transferred through at least one other department before being Discharged (in blue). This wild card event (in the middle) is represented in black.

5.3.6 Point Event Absences

Ironically, the absence of a point event actually functions as an interval of time. Consider the query for two Stroke events, with no Diagnosis in between them. The implication is that there are no Diagnosis events for the entire interval of time between the two Stroke events. This is referred to as a "span" absence: the absence of the Diagnosis event implicitly spans out to the presence events on either side of it.

Figure 5.7: Query for two stroke events (in green), with no diagnosis (in orange) in between.

In this query, the two Stroke events are represented by consecutive query elements q_y and q_{y+1}. For every Diagnosis event $e_d = \{e_{d\_t}, e_{d\_c}\}$ in the event record then, there must be a constraint specifying that e_{d_t} does not occur between the event times that are matched to q_y and q_{y+1}:

$$q_{y\_t} \geq e_{d\_t} \qquad \text{or} \qquad q_{(y+1)\_t} \leq e_{d\_t} \tag{5.8}$$

These constraints are specified by introducing a new variable $a \in [0, 1]$ and a large constant K. For this purpose K= Long.MAX_VALUE is used, which represents that highest possible time. The absence is then specified using the following

two constraints:

$$q_{y\_t} + \mathrm{K}a \geq e_{d\_t} \tag{5.9}$$

$$q_{(y+1)\_t} - \mathrm{K}(1-a) \leq e_{d\_t} \tag{5.10}$$

Using these constraints, if $q_{y\_t} \geq e_{d\_t}$, then $a = 0$ and both constraints are satisfied. Conversely, if $q_{(y+1)\_t} \leq e_{d\_t}$, then $a = 1$ and both constraints are still satisfied. Finally, if $q_{y\_t} \leq e_{d\_t} \leq q_{(y+1)\_t}$, then the constraints cannot be satisfied.

Unlike the previously discussed constraints, the number of constraints that are required to specify the absence of a point event is dependent on the number of events in the record. Specifically, it is dependent on the number of events of the absent category. While this is less desirable than the query-dependent constraints, absences are the only type of constraint that is record-dependent.

5.3.7 Interval Event Absences

The absence of an interval event can function similar to the absence of a point event. That is, it can span out to the presence events on either side of it. For example, one might formulate a query for two Stroke events, where the patient did not take his medication for the entire interval of time in between them.

Figure 5.8: Query for two stroke events (in green), with no medication intervals (in red) in between.

In this case, the absence of the interval can be specified much in the same way as the absence of a point event. For every medication interval in the patient record, represented by two event points $e_{m-start} = \{e_{m-start\_t}, e_{m-start\_c}\}$ and $e_{m-end} = \{e_{m-end\_t}, e_{m-end\_c}\}$, the interval must either end before the first Stroke event, or

begin after the second Stoke event:

$$q_{y\_t} \geq e_{m-end\_t} \qquad \text{or} \qquad q_{(y+1)\_t} \leq e_{m-start\_t} \tag{5.11}$$

However, it is also possible to query for the absence of an interval that occurs at a single point in time. This is referred to as a "point" absence. Consider, for example, the query for a patients who stopped taking their medication at some point between their two Stroke events.

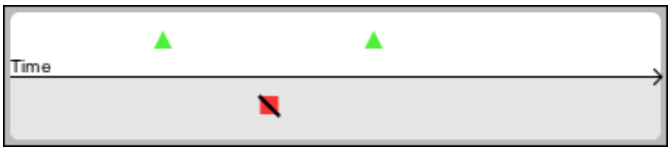


Figure 5.9: Query for patients who stopped taking their medication (in red) at some point between their two Stroke events (in green).

A "point" absence can be handled by translating it into a "span" absence. This is done by framing the absence with two wild card query elements, and allowing the absence to span out to these elements. The ordering constraints are then modified to allow for equality. That is, the wild card elements are allowed to match to the same events that are matched to the Stroke elements.

Figure 5.10: Modified query for patients who stopped taking their medication (in red) at some point between their two Stroke events (in green).

5.3.8 Flexible Ordering

One of the most salient benefits of using an integer programming approach for solving best-match event query is that ordering constraints can be added between any two events. Or perhaps better put, ordering constraints can be *not* added between any two events. Queries can be easily formulated such that certain events

can appear in any order. For example, the following query requires the occurrence of three events. However, the first two events can occur in any order.

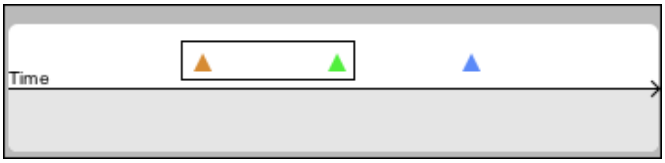


Figure 5.11: Query for three events, where the first two events can occur in any order.

Using ordering constraints, this query can be specified such that the first two events must occur before the third, but no constraint is added to dictate the ordering of the first and second event.

$$\sum_{x=1}^{i} m_{x,1} \cdot e_{x\_\mathrm{t}} < \sum_{x=1}^{i} m_{x,3} \cdot e_{x\_\mathrm{t}} \tag{5.12}$$

$$\sum_{x=1}^{i} m_{x,2} \cdot e_{x\_\mathrm{t}} < \sum_{x=1}^{i} m_{x,3} \cdot e_{x\_\mathrm{t}} \tag{5.13}$$

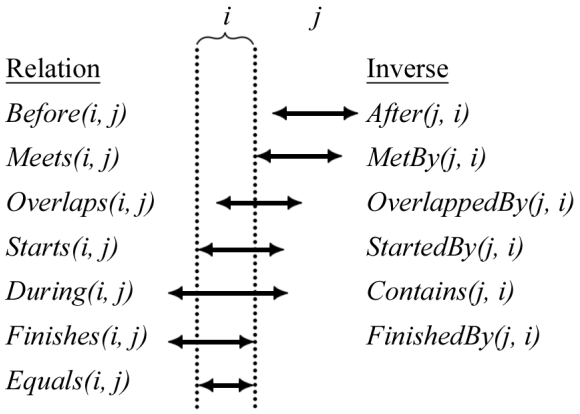


Figure 5.12: Allen's 13 interval relationships [6].

This flexibility becomes even more useful with queries involving interval events. A common query is to find records in which two intervals overlap. Allen's seminal work on interval logic [6] described 13 different ways in which two intervals can occur in relation to one another (see Figure 5.12). Of these, 8 relationships involve

an overlapping. This means that, without order independence, 8 different queries would be required to capture this basic relationship. However, an integer program can capture this relationship with only four ordering constraints, specifying only that the two interval start points must occur before the two interval end points.

$$\sum_{x=1}^{i} m_{x,1} \cdot e_{x\_t} < \sum_{x=1}^{i} m_{x,3} \cdot e_{x\_t} \tag{5.14}$$

$$\sum_{x=1}^{i} m_{x,2} \cdot e_{x\_t} < \sum_{x=1}^{i} m_{x,3} \cdot e_{x\_t} \tag{5.15}$$

$$\sum_{x=1}^{i} m_{x,1} \cdot e_{x\_t} < \sum_{x=1}^{i} m_{x,4} \cdot e_{x\_t} \tag{5.16}$$

$$\sum_{x=1}^{i} m_{x,2} \cdot e_{x\_t} < \sum_{x=1}^{i} m_{x,4} \cdot e_{x\_t} \tag{5.17}$$

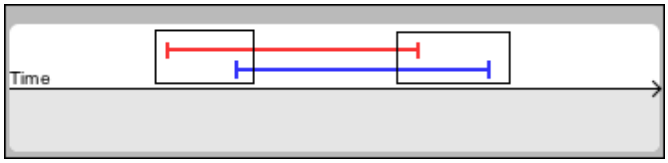


Figure 5.13: Query for overlapping intervals.

Essentially, ordering constraints must only be added when they are necessary, and each element in the query can be ordered according to any other query elements. This allows queries to incorporate a large amount of flexibility without introducing additional complexity in the query processing.

5.3.9 Event Repetition

Event repetition occurs when a given query element, or pattern of query elements, can be matched to an indeterminate number of events in the record. For example, consider the query for patients who went to the Emergency Room at least 4 times and at most 5 times after having a Stroke.

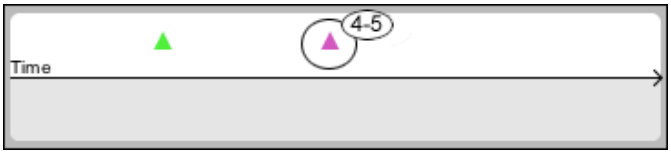


Figure 5.14: Query for patients who went to the Emergency Room (in pink) either 4 or 5 times after having a Stroke (in green).

This can be accomplished relaxing the one-to-one constraint for the final query element, allowing for this element to be matched to 0 events in the patient record (Equation 5.18). Additionally, the category constraint must be modified to account for the scenario when this element is not matched (Equation 5.19). Finally, a similar consideration must be made for the ordering constraints (again, K is used to represent a large constant - Equation 5.20).

$$\sum_{x=1}^{i} m_{x,6} <= 1 \tag{5.18}$$

$$\sum_{x=1}^{i} m_{x,6} \cdot e_{x\_c} = q_{6\_c} \cdot \sum_{x=1}^{i} m_{x,6} \tag{5.19}$$

$$\sum_{x=1}^{i} m_{x,5} \cdot e_{x\_t} < \sum_{x=1}^{i} m_{x,6} \cdot e_{x\_t} + textrmK(1 - \sum_{x=1}^{i} m_{x,6}) \tag{5.20}$$

5.3.10 The Objective Function

While the above constraint strategies will determine whether a given event record contains the query sequence, the objective function will dictate which events in the record will be matched to the query when there are multiple possible options. While the objective function is highly context dependent, the following examples should provide a sense of how the objective function can guide the selection of a best-match:

- Minimize/maximize the duration of the query sequence:

MIN: $\sum_{x=1}^{i} m_{x,j} \cdot e_{x\_\mathrm{t}} - \sum_{x=1}^{i} m_{x,1} \cdot e_{x\_\mathrm{t}}$

- Match query elements to events close to the beginning/end of the record:
  MAX: $\sum_{x=1}^{i} m_{x,j} \cdot e_{x\_\mathrm{t}}$
- Minimize/maximize the lapse in time between two elements of the query sequence:
  MIN: $\sum_{x=1}^{i} m_{x,5} \cdot e_{x\_\mathrm{t}} - \sum_{x=1}^{i} m_{x,2} \cdot e_{x\_\mathrm{t}}$
- Minimize/maximize the number of events matched to the query:
  MIN: $\sum_{x=1}^{i} \sum_{y=1}^{j} m_{x,y}$
- Minimize/maximize an attribute of one of the events:
  MAX: $\sum_{x=1}^{i} m_{x,j} \cdot e_{x\_\mathrm{a}}$

5.3.11 Non-Integer Variables

Consider the earlier example of a query for three overlapping intervals, shown again here in Figure 5.15. It would be reasonable, given a query of this type, to want to maximize the duration of the overlap.

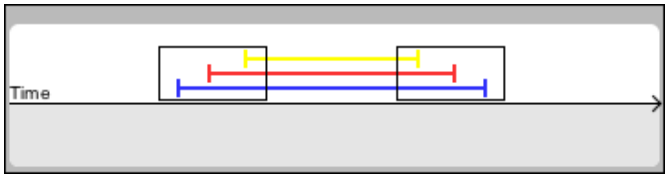


Figure 5.15: A query for three overlapping intervals, where the start and end points can occur in any order, but the duration of overlap needs to be maximized.

This can be accomplished by integrating two continuous variables, $T1$ and $T2$ into the IP formulation. $T1$ will serve to mark the start point of the interval overlapping. It is designed to take on the lowest possible time value that is still

greater than the timestamps of the three matching interval start-points. This is done with the three following constraints:

$$\sum_{x=1}^{i} m_{x,1} \cdot e_{x\_\mathrm{t}} < T1 \tag{5.21}$$

$$\sum_{x=1}^{i} m_{x,2} \cdot e_{x\_\mathrm{t}} < T1 \tag{5.22}$$

$$\sum_{x=1}^{i} m_{x,3} \cdot e_{x\_\mathrm{t}} < T1 \tag{5.23}$$

$T2$ then, will take on the highest possible time value that occurs before the timestamps of the three matching interval end-points. Again, this is accomplished using the following three constraints:

$$\sum_{x=1}^{i} m_{x,4} \cdot e_{x\_\mathrm{t}} > T2 \tag{5.24}$$

$$\sum_{x=1}^{i} m_{x,5} \cdot e_{x\_\mathrm{t}} > T2 \tag{5.25}$$

$$\sum_{x=1}^{i} m_{x,6} \cdot e_{x\_\mathrm{t}} > T2 \tag{5.26}$$

While the above constraints will ensure that $T1$ and $T2$ are sequenced properly, the objective function is what will push those values out to their respective minimum and maximum. This is accomplished by specifying the objective function to maximize the difference between these two variables:

$$\text{MAX: } T2 - T1 \tag{5.27}$$

5.4 An Example

Consider the following query, being evaluated against the given event record. In this case, we are looking for patients who had never had a Stroke prior to taking Drug A, but then had a Stroke after starting their prescription. We would like to ensure that the patient took the prescription for at least one month, and finally, we would like to minimize the time lapse between the start of the prescription and the Stroke. The query evaluation process begins by breaking the query and the event record into their respective sets $Q = \{q_1, q_2, q_3\}$ and $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ (Figure 5.16).

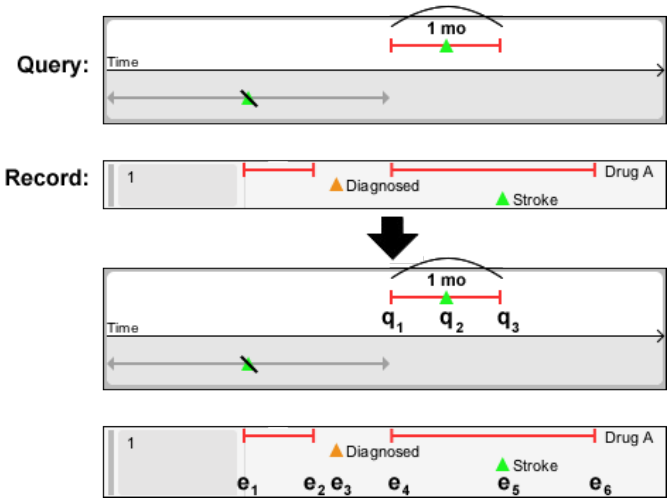


Figure 5.16: The query and the event record get broken into sets $Q = \{q_1, q_2, q_3\}$ and $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ respectively.

For the sake of clarity in the example, event timestamps and categories will be represented as integers. To distinguish between interval start and end points, which have the same event category but cannot be matched to the same events, interval end points will be represented by appending a decimal to the original category. The sets Q and E, and the match table are shown in tables 5.2 and 5.3 respectively.

$q_y = \{q_{y\_c}\}$	$e_x = \{e_{x\_t}, e_{x\_c}\}$
$q_1 = \{1\}$	$e_1 = \{1, 1\}$
$q_2 = \{2\}$	$e_2 = \{3, 1.5\}$
$q_3 = \{1.5\}$	$e_3 = \{4, 3\}$
	$e_4 = \{6, 1\}$
	$e_5 = \{7, 2\}$
	$e_6 = \{9, 1.5\}$

Table 5.2: Sets Q (left) and E (right).

	q_1	q_2	q_3
e_1	$m_{1,1}$	$m_{1,2}$	$m_{1,3}$
e_2	$m_{2,1}$	$m_{2,2}$	$m_{1,1}$
e_3	$m_{3,1}$	$m_{3,2}$	$m_{3,3}$
e_4	$m_{4,1}$	$m_{4,2}$	$m_{4,3}$
e_5	$m_{5,1}$	$m_{5,2}$	$m_{5,3}$
e_6	$m_{6,1}$	$m_{6,2}$	$m_{4,1}$

Table 5.3: Binary variables of the match table.

The IP is then formulated using the 9 constraints and objective function depicted in Figure 5.17. The one-to-one constraints are enforced by equations 5.28-5.28. Category constraints are enforced by equations 5.28-5.28, and ordering is maintained by equations 5.28 and 5.28 (note that the Stroke event and the prescription end point can occur in any order). The absence of a Stroke event prior to the prescription is handled by equation 5.28. Finally, the objective function is set by equation 5.28.

Hopefully it is clear that, intuitively, the given event record should match to this query based on the following event matches: $q_1 \rightarrow e_4$, $q_2 \rightarrow e_5$, $q_3 \rightarrow e_6$ (Figure 5.18), which corresponds to the match table shown in Table 5.4. These values can

$$m_{1,1} + m_{2,1} + m_{3,1} + m_{4,1} + m_{5,1} + m_{6,1} = 1$$
$$m_{1,2} + m_{2,2} + m_{3,2} + m_{4,2} + m_{5,2} + m_{6,2} = 1$$
$$m_{1,3} + m_{1,1} + m_{3,3} + m_{4,3} + m_{5,3} + m_{4,1} = 1$$
$$m_{1,1} + 1.5m_{2,1} + 3m_{3,1} + m_{4,1} + 2m_{5,1} + 1.5m_{6,1} = 1$$
$$m_{1,2} + 1.5m_{2,2} + 3m_{3,2} + m_{4,2} + 2m_{5,2} + 1.5m_{6,2} = 2$$
$$m_{1,3} + 1.5m_{1,1} + 3m_{3,3} + m_{4,3} + 2m_{5,3} + 1.5m_{4,1} = 1.5$$
$$m_{1,1} + 3m_{2,1} + 4m_{3,1} + 6m_{4,1} + 7m_{5,1} + 9m_{6,1} < m_{1,2} + 3m_{2,2} + 4m_{3,2} + 6m_{4,2} + 7m_{5,2} + 9m_{6,2}$$
$$m_{1,1} + 3m_{2,1} + 4m_{3,1} + 6m_{4,1} + 7m_{5,1} + 9m_{6,1} + 1 < m_{1,3} + 3m_{2,3} + 4m_{3,3} + 6m_{4,3} + 7m_{5,3} + 9m_{6,3}$$
$$10 > m_{1,1} + 3m_{2,1} + 4m_{3,1} + 6m_{4,1} + 7m_{5,1} + 9m_{6,1}$$
$$\text{MIN: } m_{1,3} + 3m_{1,1} + 4m_{3,3} + 6m_{4,3} + 7m_{5,3} + 9m_{4,1} - m_{1,2} - 3m_{2,2} - 4m_{3,2} - 6m_{4,2} - 7m_{5,2} - 9m_{6,2}$$

Figure 5.17: The 9 constraints of the IP, followed by the objective function (Equation 5.28).

be plugged back into the 9 constraints to confirm that they all hold.

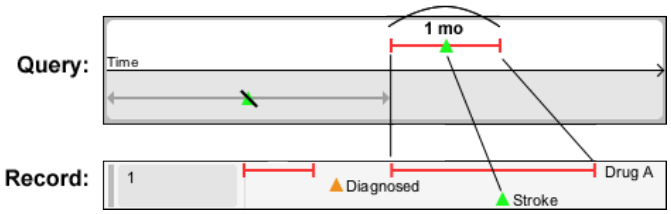


Figure 5.18: The query and the event record are a match given $q_1 \rightarrow e_4$, $q_2 \rightarrow e_5$, $q_3 \rightarrow e_6$.

	q_1	q_2	q_3
e_1	$m_{1,1} = 0$	$m_{1,2} = 0$	$m_{1,3} = 0$
e_2	$m_{2,1} = 0$	$m_{2,2} = 0$	$m_{1,1} = 0$
e_3	$m_{3,1} = 0$	$m_{3,2} = 0$	$m_{3,3} = 0$
e_4	$m_{4,1} = 1$	$m_{4,2} = 0$	$m_{4,3} = 0$
e_5	$m_{5,1} = 0$	$m_{5,2} = 1$	$m_{5,3} = 0$
e_6	$m_{6,1} = 0$	$m_{6,2} = 0$	$m_{4,1} = 1$

Table 5.4: The final match table values.

5.5 Query Scope Extensions

Though this integer programming back-end was intended to merely replicate the query scope described in Chapter 4, it presents an opportunity to easily implement additional shortcuts that would not have been as easily implemented using the scan-and-backtrack processing method. As discussed in Section 4.3.4, shortcuts do not expand the scope of the query system, but allow users to capture multiple event sequences in a single query specification. In this case, the integer programming back-end makes it very easy for users to combine unrelated components into a single query. Consider, for example, a query for two medications that must overlap, and then two symptoms that must occur after the start of the combined dosage. In this case, the two symptom events do not have any implied relationship to the two

interval end-points (see Figure 5.19).

Accommodating a shortcut like this would require significant modifications to the scan-and-backtrack processing method. However, using the integer programming back-end, this shortcut can easily be supported by simply not specifying certain ordering constraints. In this case, the overlapping intervals could be captured using the same constraints as described in Section 5.3.8. From there, only three additional ordering constraints are needed: one that specifies that the first symptom must occur after the start of the first medication, one that specifies that the first symptom must occur after the start of the second medication, and one that specifies that the second symptom must occur after the first symptom. Ordering constraints that are not necessary, such as any ordering constraint between the symptoms and the medication end-points, are simply not specified.

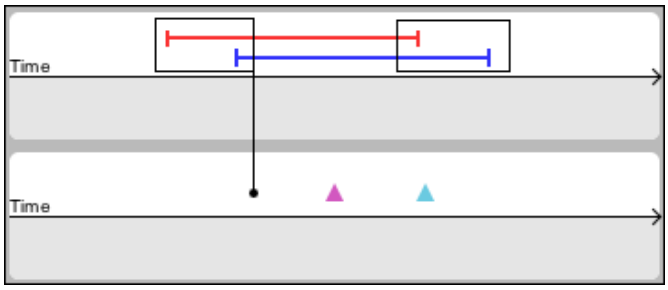


Figure 5.19: Query for two events that must occur after the start of an overlap.

5.6 Initial Experiments

To evaluate the feasibility of IP query processing in practice, three representative sample queries of increasing complexity (Q1, Q2, and Q3 - shown in Figures 5.20, 5.21, and 5.22, respectively) were run against four different record sizes: 20 events, 40 events, 80 events, and 160 events. The queries were run using the open source (mixed-integer) linear programming solver, `lp_solve` [7]. This solver was chosen for its accessibility, and its well-documented implementation. However, `lp_solve` is only one of many open-source and commercial linear optimization solvers that

could potentially be used to solve these queries.

Each query was run multiple times against both a matching and a non-matching record at each record size. The run times are depicted in Figures 5.23 and 5.24. Again, the goal of these experiments was to evaluate whether an IP processing approach would even be feasible in order to be considered as a potential back-end for the EventFlow query system. The experiments were meant to target broad aspects such as whether the query completed, completed accurately, and completed within a reasonable time frame.

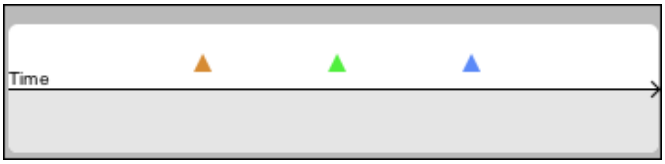


Figure 5.20: Q1 - Test query for three sequential point events.

Figure 5.21: Q2 - Test query incorporating both absences and repetition.

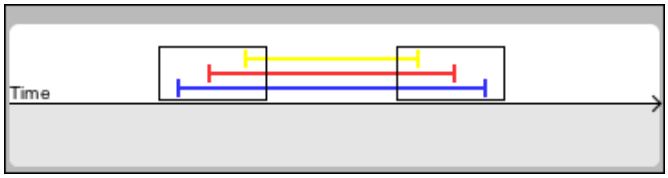


Figure 5.22: Q3 - Test query for interval overlapping.

The three queries were chosen for their ability to push the limits of `lp_solve` in different ways as they scaled up to larger event records. Q1 is the simplest query in that it is the easiest to match. That is, an event record with any permutation of these three event types will likely be able to match this subsequence of events. However, as the record gets larger, the number of possible ways to match the query increases exponentially, requiring the system to choose the best match out of a large number of possible options.

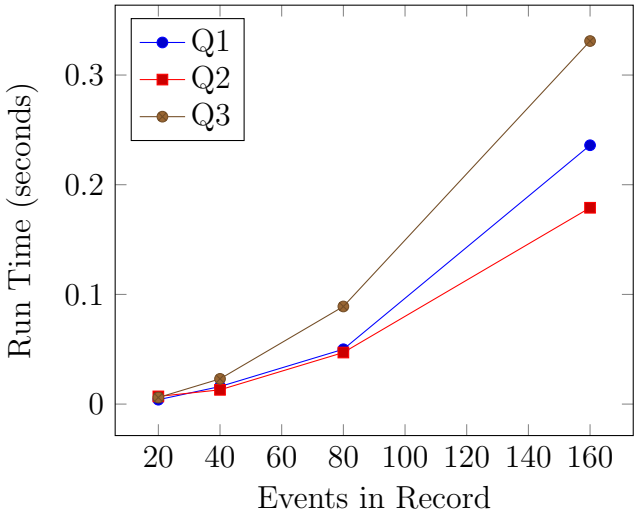


Figure 5.23: Run times against matching records.

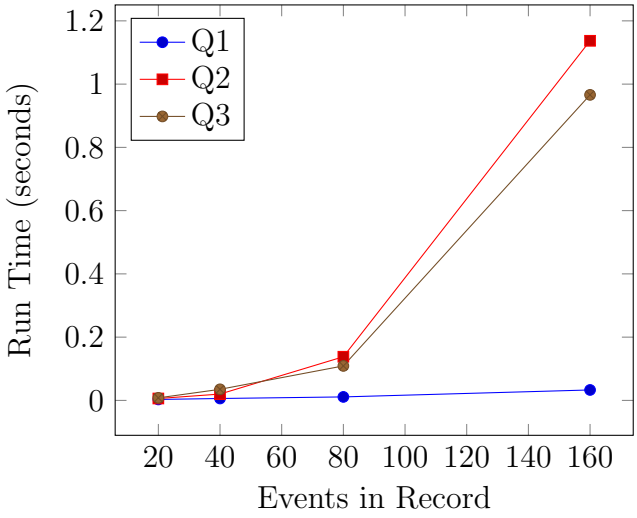


Figure 5.24: Run times against non-matching records.

Q2 was designed to stretch the number of constraints that comprise the IP. Due to the absence element in the query, two constraints must be added for each record event of that category. The number of absence constraint pairs increased from 7 to 63 as the record size increased from 20 events to 160 events.

Finally, Q3 was chosen to represent the hardness of the interval subgraph isomorphism problem. This query is not only the most complex, but is rarely ad-

dressed without the use of custom preprocessing and data structures.

As would be expected, the run times of `lp_solve` scaled exponentially with all three queries as the record sizes increased. However, the testing revealed three interesting phenomenon:

1. Running the IP against matching records scaled better than running the IP against non-matching records. For Q3, the non-matching record was 3 times slower. For Q2, it was 6 times slower.

2. Q1 was the exception to this observation, indicating that the nature of the non-match may have a noticeable effect on the resulting run time.

3. The large number of constraints in Q2 had no noticeable effect on the run time. In fact, the run time for Q2 against a matching record was faster than both Q1 and Q3.

Overall, the initial testing revealed that `lp_solve` was able to accurately arrive at a solution in all cases. Furthermore, while the run times did appear to scale exponentially, this did not produce a dramatic effect until around 80 events per record. In the 20-40 events per record range, which is much closer to the number of events per record at which the EventFlow visualization in designed to operate, run times were clustered into a fairly predictable range. This range was capped by the 40ms mark, further indicating that this approach was not obviously prohibitively slow. The initial testing of the IP back-end, though limited in scope, was sufficient in indicating that the method was valid, and that it had the potential to be competitive with EventFlow's scan-and-backtrack processing implementation.

5.7 Further Experiments and Discussion

Based on the initial testing of the IP back-end, further testing was conducted to better evaluate the run times of `lp_solve`, and to more directly compare this

strategy to EventFlow's scan-and-backtrack processing implementation. To do this, a representative set of 10 queries, shown in Figure 5.25 were run against the LABA dataset (see Section 7.2). This dataset was chosen to provide a realistic amount of intra-record variation involving both point and interval events. The dataset consisted of 100 records, most of which contained a unique sequence of events.

This intra-record variety was the paramount focus of testing, since the IP back-end scales exponentially within records, rather than between them. That is, the performance of the IP back-end degrades as the number of query elements or the number of events within a single record increases. However, since each record is solved as a separate integer program, the number of records in the dataset does not factor into this exponential scaling. Datasets of 1,000 or even 10,000 records will, at worst, scale linearly, but could also be processed in parallel. In either case, the average time to process a single record will ultimately dictate performance.

Since EventFlow's scan-and-backtrack query processing was only designed to identify the first matching event sequence in a given record, the queries passed to `lp_solve` were formulated with an objective function that attempted to minimize the time of the last matched query element. As a result, both systems were searching for the same event sequence. While this allows for a more direct comparison, it does not fully showcase the extended capabilities of the IP back-end to match events according to any optimization function. Additionally, the run times reported in this section pertain only to the query execution. This does not account for the custom data structures and preprocessing that EventFlow's scan-and-backtrack implementation relies upon. The time required to construct and organize these structures were inextricably linked to other components of the application, and thus could not be isolated.

It is also critical to point out that, while `lp_solve` was the most readily available integer program solver, commercial IP solvers such as Cplex and Gurobi offer significant performance gains over `lp_solve`. In a direct comparison of IP

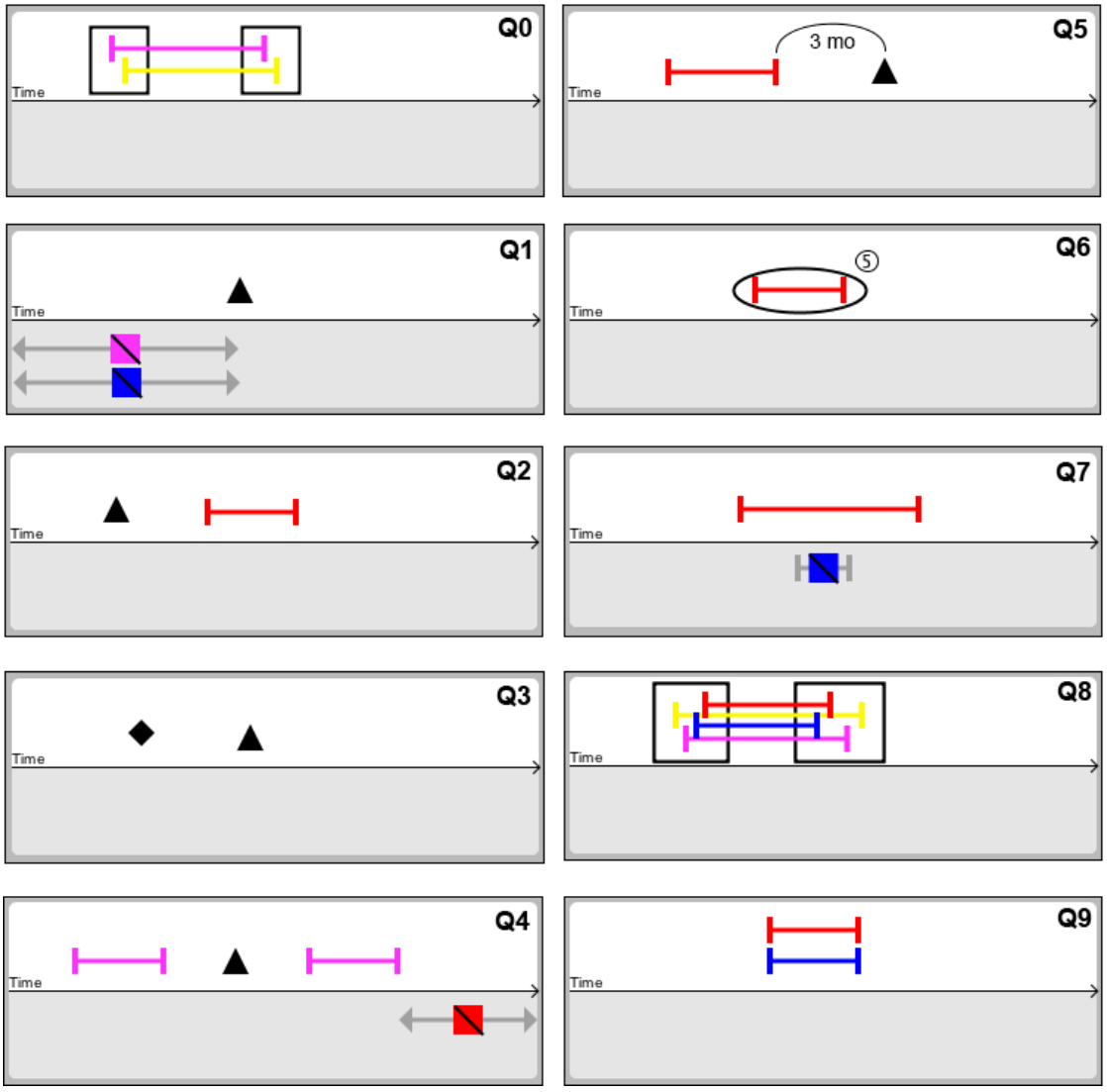


Figure 5.25: Ten different queries, all relevant to the LABA case study, were processed using both EventFlow's scan-and-backtrack back-end and `lp_solve`'s integer programming back-end.

solvers, Meindl and Templ used well-established benchmarks to observe that both Cplex and Gurobi outperform `lp_solve` by a factor of between 15 and 20 [68]. These findings can be used to estimate the performance of a commercial solver based on the run times of `lp_solve`. The run times, averaged over five different runs of each query, are listed below in Table 5.26.

Query	`lp_solve`	Projected IP Time	Scan-and-Backtrack
Q0	.384	.027	.023
Q1	.176	.012	.015
Q2	.174	.011	.030
Q3	.164	.012	.016
Q4	.249	.017	.011
Q5	.177	.012	.012
Q6	.621	.041	.044
Q7	.261	.017	.026
Q8	1.763	.117	.062
Q9	.282	.019	.034

Figure 5.26: The averaged run times (in seconds) of each query using `lp_solve` and EventFlow's scan-and-backtrack processing. The run times of `lp_solve` were reduced by a factor of 15 to estimate the run times of a commercial IP solver.

Though the run times of `lp_solve` were significantly higher than those of EventFlow's scan-and-backtrack implementation, the run times of a commercial IP solver can be estimated by reducing the run times of `lp_solve` by a factor of 15. After this reduction, the projected run times of a commercial solver *are* competitive withe the scan-and-backtrack back-end. The only notable exception is Q8 which, even after the reduction, still ran at half the speed of the scan-and-backtrack back-end. This is not surprising as Q8 is the most complex query of the set, and while the scan-and-backtrack method need only find the first instance of this pattern before terminating, the IP back-end must consider all possible matches before optimizing for the lowest end-time. It is worth noting, however, that with a one-line change to to IP formulation, `lp_solve` could find the longest duration of overlap between these four event categories. Accommodating this objective in the scan-and-backtrack implementation would not only require a significant reworking of the underlying algorithm, but would dramatically increase the run time.

Also worth considering are the various ways that the IP processing method could be further optimized. During the testing reported here, `lp_solve` was passed the entire event record for every record in the dataset. However, this set could be reduced, either by including only the event categories that are relevant to the query, or by performing some simple tests upfront to prevent obvious non-matches from being attempted by the IP solver. Additionally, event record simplifications such as the interval merging (see Chapter 6), which requires only a single scan through each record, could dramatically reduce the number of events that are passed to the IP solver.

5.8 Summary

In this chapter, I demonstrated that best-match temporal event queries can be formulated and solved as integer programs. This approach was motivated by the innate complexity of these queries, and offers tangible advantages over other approaches in that it does not require the events in the underlying record to be sorted, it provides a natural ranking metric for the results, and it can be both constructed and altered incrementally as the query requirements change.

Though solving integer programs is NP-Hard, I showed that certain types of temporal event queries are NP-Hard as well, and thus an IP processing strategy should not be immediately ruled out. Testing revealed that open-source IP solvers could successfully and consistently solve IP formulations of temporal event queries. Using these run times, and given that EventFlow typically operates on records consisting of between 20 and 40 events, it was projected that a commercial IP solvers would be competitive with EventFlow's current scan-and-backtrack query processing.

Ultimately, the ideal back-end processing strategy depends on the needs and objectives of the overall query system. If the query system need only handle a simple set of event patterns, and is only trying to determine if that pattern ex-

ists in a given record, the scan-and-backtrack implementation is easily sufficient. However, if ranking and best-match query are primary objectives, then a commercial IP solver could cover a wide range of both ranking metrics and complex query specifications without incurring significantly great run time costs. From an implementation perspective, the IP formulations are both easy to construct as well as easy to extend with additional query features. If record sizes remain bounded, which is already a requirement of the EventFlow visualization, then IP query processing presents as a tangible solution.

Chapter 6

Event Sequence Transformation and Simplification

Electronic Health Records (EHRs) have emerged as a cost-effective data source for conducting medical research. Across the world, sensors and forms and spreadsheets capture and store every aspect of medical treatment as sequences of categorized, timestamped events. The motivating question is simple: how can we best leverage the things that have already happened to inform our future actions? To this end, researchers approach these datasets with the goal of gaining understanding on two different levels:

- **Intra-Record Understanding:** Knowledge is gained about the sequence of events that comprises a single record (i.e. a patient record or a shipment record): Why was this patient transferred from the Emergency Room to the Intensive Care Unit (ICU)?
- **Inter-Record Understanding:** Knowledge is gained about population-level trends and patterns across an entire group of records: Of all the patients who were transferred to the ICU, what was the most common outcome?

Three years ago, when LifeFlow caught the attention of researchers at the U.S. Army Pharmacovigilance Center (PVC), they were conducting a long-term study on asthma treatment and the prescription of Long-Acting Beta Agonists (LABAs). This type of medication should only be prescribed when alternate treatments have proven ineffective. It is meant to be both preceded and followed by prescriptions for other, less-potent asthma medications [8]. The question researchers were

trying to answer, given a sample dataset of 100 patients, was whether this recommended practice was actually being followed. Over my initial year of working with the PVC, LifeFlow was redesigned and restructured to accommodate their interval-based datasets (see Chapter 3). The PVC was given the updated software, redubbed as EventFlow (see Figure 6.2), and my work, it would seem, was finished.

Then the PVC researchers loaded their LABA dataset...

The resulting display, shown in Figure 6.1-left, was so visually complex that it was initially described as "confetti." EventFlow's aggregated display is generated by grouping records with the same event sequence, however, the patient records in the LABA dataset were so long and varied that the chances of two records having the same sequence were extremely small. In this case, EventFlow cannot effectively group records, and the aggregated display defaults to rendering each record individually. The ability to leverage the visualization for inter-record understanding is lost.

This visual complexity became less of an isolated incident and more of a theme as I took on additional case studies in the medical domain. The complexity of raw EHR data manifests itself on both the intra-record and inter-record fronts. For example, it is not always clear which symptoms motivate which treatments. Relationships between events have to be inferred from the raw data alone. A single patient record may contain simultaneous treatments for multiple conditions (intra-record complexity). Additionally, EHR data is collected under real-world conditions, where variables cannot be meticulously controlled. The strategy for boosting the signal through this noise then, is to use increasingly larger datasets (inter-record complexity).

Despite the size and heterogeneity of EHR datasets, however, the questions being asked of them typically only revolve around a subset of patients and events. Furthermore, due to the nuances of data entry and extraction, the event sequences

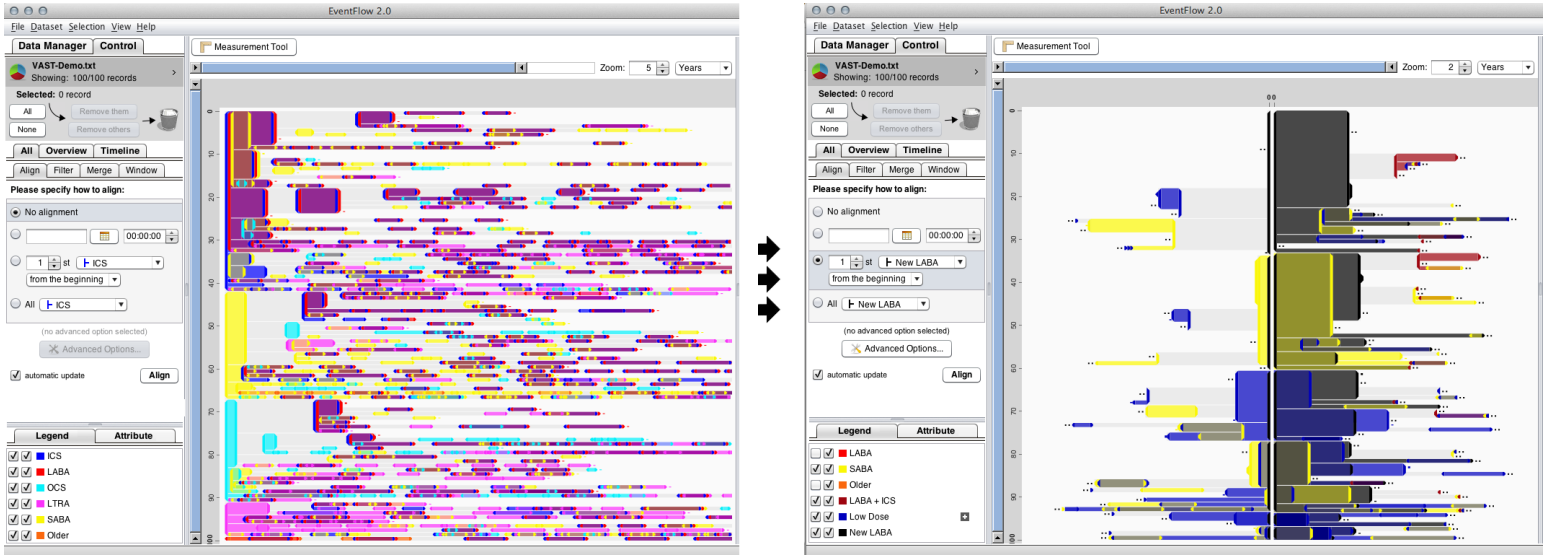


Figure 6.1: In EventFlow, the original LABA dataset, consisting of over 2700 visual elements (left), was quickly pared down to the events most critical to the study. The simplified dataset (right) consists of only 492 visual elements, an 80% reduction in visual complexity. From this simplified figure, aligned by the patients' "new" LABA prescription, researchers were immediately able to notice the data sparsity on the left side of the alignment point, indicating that patients had not received other treatments in the months leading up to their LABA prescription (i.e. not following the recommended practices).

in an initial dataset are frequently *more* complex than the real-world events that they represent. There is a huge opportunity to discard irrelevant data points, bearing in mind that relevance changes on a study-by-study or even a question-by-question basis. This chapter describes the design of a series of intuitive and effective controls that allow users to quickly transform and simplify an event record dataset down to its most accurate representation and its most meaningful elements. My process was structured around the design study methodology described in [95]. Not surprisingly, the effort focused on two different strategies for transforming and simplifying a dataset:

- **Intra-Record Transformation/Simplification:** Altering the events within an event record.
- **Inter-Record Simplification:** Altering the subset of records in the dataset.

These strategies were evaluated in multiple long-term case studies, many of which are described in additional detail in Chapter 7:

- **LABA Study** with the U.S. Army Pharmacovigilance Center.
- **Opioid Misclassification** with the U.S. Army Pharmacovigilance Center.
- **The Diabetic Foot** with the University of Florida College of Medicine.
- **Attention Deficit Disorder Treatment** with the University of Maryland School of Medicine.
- **Pediatric Trauma Procedures** with MedStar Health Facilities.

The LABA study will serve as an overarching example throughout this chapter. The Opioid Misclassification study and the Pediatric Trauma Procedures study will be presented, in greater detail, in Sections 6.5 and 6.6. I also describe the use of transformation and simplification on an experimental dataset of basketball statistics to demonstrate the generalizability of these approaches. Across

all of these studies, I found that complex datasets could consistently be simplified by around 80%, and that extraneous records and events could be both removed and re-inserted in only a few clicks. I also found that event sequences could be transformed to better reflect both reality and the perspective of the study at hand. The resulting visualizations allowed researchers to generate novel insights through both hypothesis generation and hypothesis testing, and communicate these results to their peers in clean, readable figures.

These new transformations required extensive software engineering efforts with novel data structures, but more importantly, they required fresh ways of thinking both for users and as a designer. This involved defining a clear language of simplification, as well as metrics for evaluating its success. The filter and transformation-based simplifications discussed in this Chapter, and the open-ended possibilities of a universal Find & Replace system of transformation and simplification, dramatically expand the potential to extract signal from noise in large datasets.

6.1 Complexity and Aggregation

It is impossible to address simplification without first defining some notion of complexity. Unfortunately, there is no single, universal metric for defining and measuring visual complexity. The topic has been extensively discussed in psychology [28, 99, 112] and, more recently, computer science [31, 60, 120]. The lack of a consensus suggests that the scale of visual complexity depends, to some extent, on the style and structure of the visualization itself.

To further complicate matters, the complexity of the visualization becomes irrelevant if the dataset has been simplified beyond use. This is a fine line to walk when the visualization is being used for both hypothesis testing and hypothesis generation. For hypothesis testing, there is a specific question to be answered, and users typically have a clear idea of what events and records are relevant to that question. In this case, the visualization can be simplified until only those elements

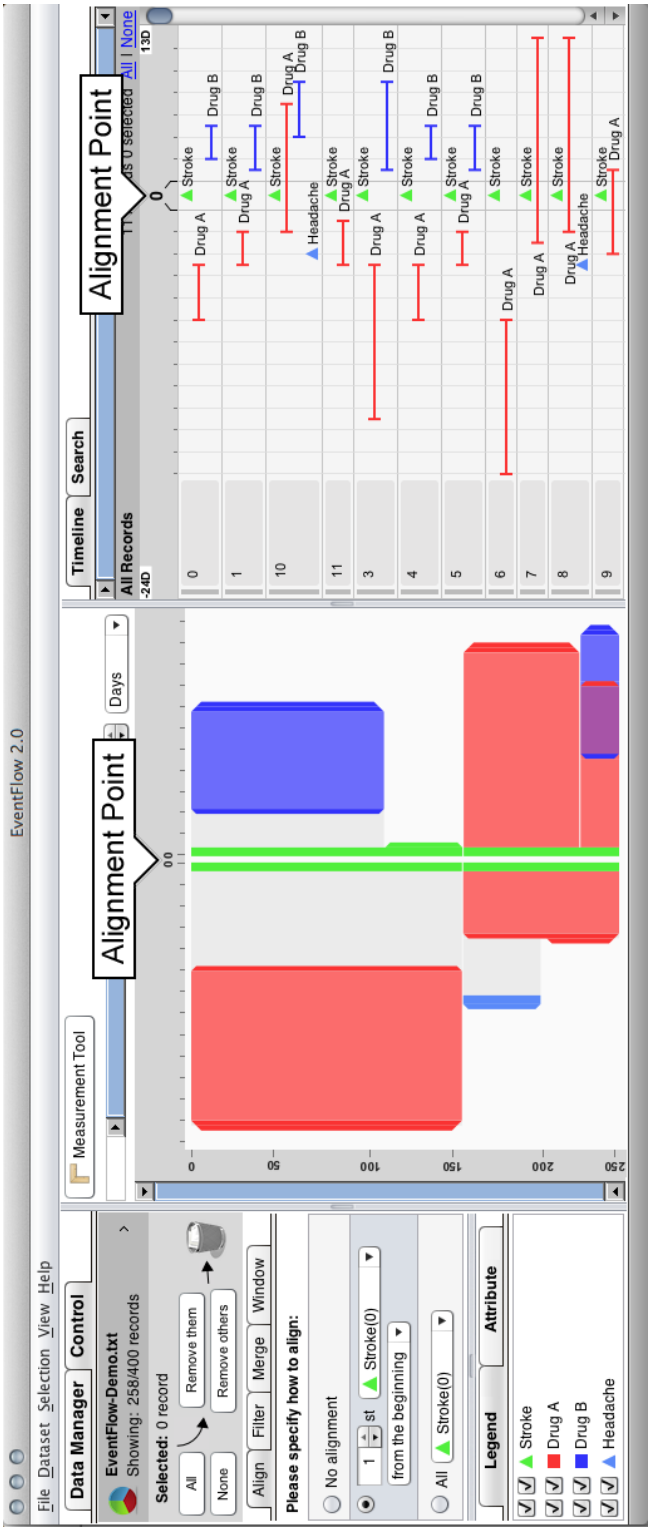


Figure 6.2: EventFlow's three panels: the control panel (left), the aggregated record display (center), and the individual record display (right). Here, a sample dataset is aligned by the Stroke event in each patient record, creating mirrored alignments in both the individual and aggregated displays. In EventFlow, record filtering is done using the "Remove" buttons at the top of the control panel. Category filtering is done using the checkboxes in the legend. Time and Attribute filtering is done using the "Window" and "Attribute" tabs respectively.

remain. Hypothesis generation, however, is a more open-ended process. Users run the risk of oversimplifying their data and missing potential insights. Thus, while complexity metrics are presented here and utilized in the following sections, it should be emphasized that these metrics must be tempered with some subjective notion of maintaining the integrity of the dataset.

As mentioned previously, EventFlow creates an aggregated view of a dataset by grouping records with the same event sequence. This grouping is done using a tree-structure that branches as the event sequences diverge from each other (see [123] for a detailed description this process). Events are represented using vertical bars, where the height of each bar is dictated by the number of records grouped at that branch. The default behavior is to root the aggregation at the beginning of each record, however users can also root the aggregation at any alignment point of interest [118]. For example, in the LABA study, patients were frequently aligned by their first LABA prescription. In this case, two aggregations are done: one for the events that led up to this alignment point, and one for those that followed it (see Figure 6.1-right).

Because of this aggregation strategy, metrics such as the number of patient records, or even the number of total events in the dataset are not reliable predictors for the complexity of the resulting visualization. For example, Figure 6.2 depicts over 250 patient records in a clean and easily readable display. Ultimately, the complexity of EventFlow's visualization is dictated by how frequently, and the extent to which, the records in the dataset share the same event sequence. In medical datasets, this can be a rare occurrence [79].

As such, visual complexity will be defined and measured here by two different metrics: the number of visual elements in the display, and the average size of these elements. The first metric, proposed in [31] and further supported in work by Rosenholtz et al. [90], is simply a count of each vertical, colored bar in EventFlow's aggregated display. This metric gives a sense of how difficult it will be for users to weave the unique elements in the display into a holistic impression of their dataset.

The number of visual elements should *decrease* as the dataset is simplified.

The second metric, defined more specifically as the average height of each vertical bar (as a percentage of the display height), provides insight into how well the individual records are being aggregated together into a summarizing display. Though this novel metric is specific to EventFlow, it draws on two of the oldest notions of visual complexity: separability and information density. Separability, introduced by Garner in 1974 [34], describes how larger items are easier to distinguish from one another, thus reducing the perceived complexity. Information density, proposed by Edward Tufte [109], relates to the amount of information conveyed by each visual element. In EventFlow, the height of each vertical bar represents the number of records aggregated at that branch. Thus, the average height across all these elements should *increase* as the dataset is simplified.

The initial LABA dataset began with over 2700 visual elements, each averaging only 1.14% of the total display height (Figure 6.1-left). From this display, the PVC researchers found it virtually impossible to visually parse out any meaningful trends. The following three sections describe the simplifications that these researchers used to ultimately gain novel insight from this dataset using EventFlow.

6.2 Filter-Based Simplifications

Filter-based simplifications allow users to remove events and records from the dataset based on the fundamental features of the data type. They have been implemented, in some form, across a wide range of visualization and command-based tools and are an essential component of visual analytics. I discuss them here for the sake of completeness, but more importantly, to demonstrate that in real-world data analysis, filter-based simplifications are most effective when they can be interleaved with more advanced simplifications. In EventFlow, filter-based simplifications are designed to be readily accessible and easily reversible (see Figure 6.2).

The use of EventFlow's filter-based simplifications can be demonstrated by an initial question that the PVC researchers had while reviewing the LABA dataset. The dataset was set up such that one LABA prescription in each patient record had been flagged with an attribute to indicate that this was the patient's "first" LABA prescription. It was around this prescription that they planned to look for the intended pattern of less-potent asthma medications.

To make this attribute visible in EventFlow's aggregated display, the researchers converted it into a marker event that was inserted into each record (see the full description of marker events in Section 6.3). When the dataset was then aligned by this marker, they noticed that, for many patients, their so-called "first" LABA prescription was actually preceded by other LABA prescriptions.

The researchers determined that this was occurring because the criteria used for identifying the "first" LABA prescription was only contingent on there being no other LABA prescriptions in the preceding 3 months. However, the original data extraction was done by selecting all prescriptions for the entire year preceding the "first" LABA, resulting in the inclusion of LABA prescriptions that were greater than 3 but less than 12 months prior. Their question was whether this inconsistency would be eliminated if they updated the extraction script to require that the "first" LABA be preceded by 6 months without another LABA prescription.

By Record

Filtering by record allows users to remove a subset of records, either through query (described in further detail in Section 6.4), or by simply clicking any element in the individual or aggregated display. Removing a large subset of uninteresting records can free up a substantial amount of screen space to render the interesting elements at a larger scale. For example, out of the entire LABA dataset, the question of whether or not it would be necessary to do a new data extraction could be answered by examining the relationship between only two event points in a subset of the patient records. To isolate the critical records, the researchers used

EventFlow's query system (Chapter 4) to identify patients whose "first" LABA prescription was preceded by another LABA prescription. Records not matching this search were filtered from the dataset, narrowing 100 records down to 26 (see Figure 6.3, Steps 1 → 2).

Step 1: 2724 elements, 1.14% of display height per element

Step 2: 993 elements, 4.69% of display height per element

By Category

EventFlow's most fundamental form of intra-record simplification is the ability to remove event categories from the display using the category check boxes in EventFlow's legend. Reducing the number of event categories can significantly increase the chances that two records will have the same event sequence and thus aggregate into fewer and larger visual elements. Returning to the question of whether to perform a new data extraction, the PVC researchers narrowed the LABA dataset down further by filtering out all of the event categories except LABA and "first" LABA events (see Step 3 of Figure 6.3). Some of these filtered categories, in fact, remained excluded from the dataset for the duration of the study.

Step 3: 99 elements, 15.01% of display height per element

By Time

Filtering by time allows users to limit EventFlow's aggregated display to a certain window of time around the current alignment point. Since the chances of two records having the same event sequence decreases as you get further and further away from the alignment point, the visual elements at the far ends of the display will be the smallest. The time filter allows users to limit the display to only the largest, most information dense elements.

The final step in simplifying the LABA dataset to answer the "first" LABA

question was to filter out all events more than 6 months prior to the alignment point. In doing this, researchers could clearly see that only one patient had their *actual* first LABA prescription within 6 months of their "first" LABA event. All of the other patients had LABA prescriptions prior to this cutoff point. This meant that only one patient would have been excluded from the dataset if the extraction were changed to require that the "first" LABA be preceded by 6 months without another LABA prescription. Based on this information, the PVC researchers decided that it was not worth doing a new data extraction. The "first" LABA prescription, from this point on, was referred to as a "new" LABA prescription. This simplification process can be seen in its entirety in Figure 6.3.

Step 4: 35 elements, 22.75% of display height per element

By Attribute

EventFlow allows attributes to be assigned to both records and individual events. For example, a patient record might have the attribute "gender," and a prescription event might have the attribute "dosage." Users can perform both intra-record simplification and inter-record simplification by removing events or records based on their attributes. Again, this serves to either increase the chances of aggregation, or free up screen space for the records of interest. While this filtering technique was not used as part of the LABA study, it was used extensively on many of the other datasets.

6.3 Transformation-Based Simplifications

While the filter-based simplifications in the previous section can significantly reduce large datasets, they are designed only to *remove* information. However, real-world datasets are not only messy with respect to volume, but also with respect to the logical translation between the event data and the real-world events that

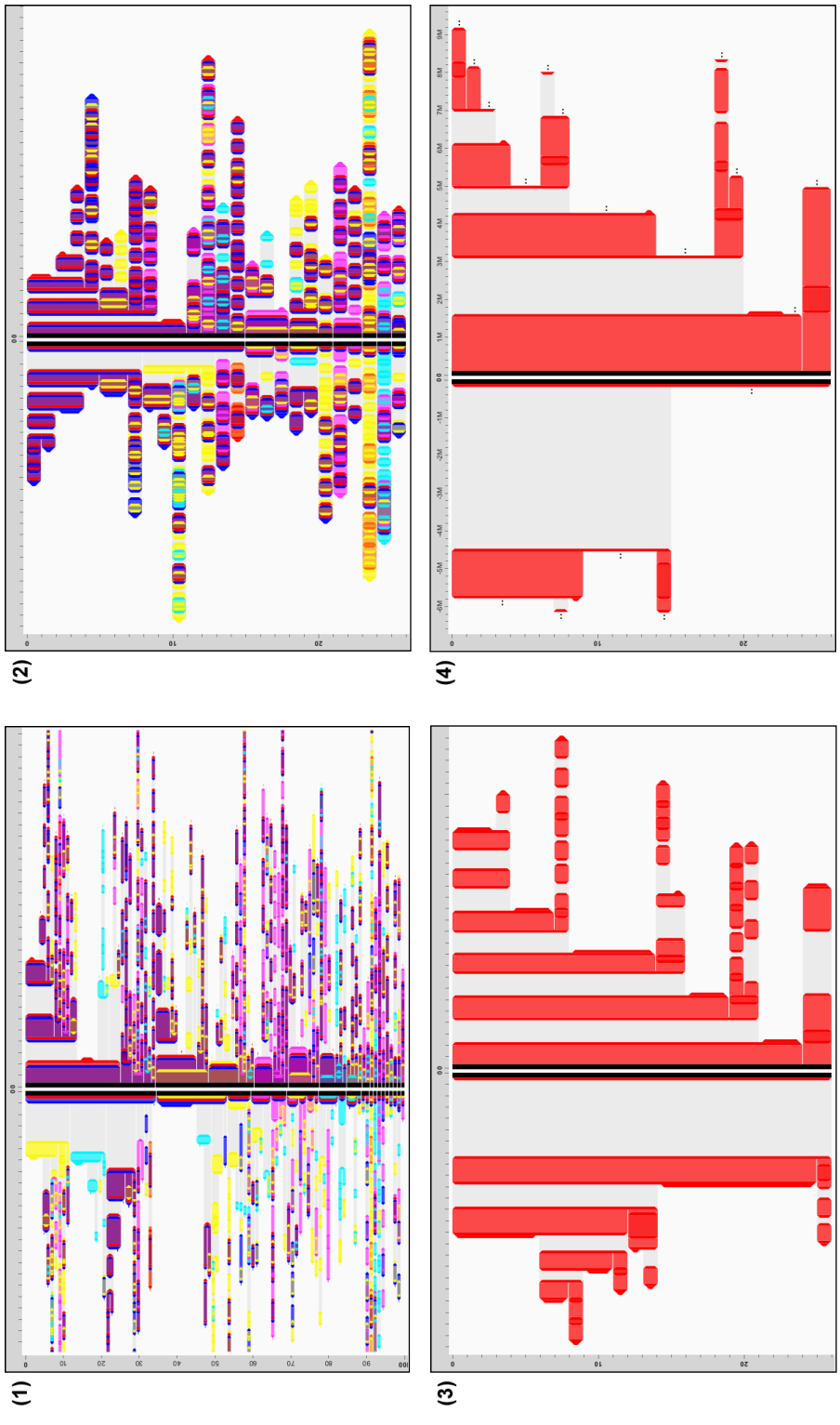


Figure 6.3: Should a new data extraction be done to exclude patients who had LABA prescriptions prior to their "first" LABA prescription? - **Step 1:** The original LABA dataset is aligned by the "first" LABA marker event (in black). **Step 2:** Patients without a LABA prescription before their "first" LABA are filtered out. **Step 3:** Categories other than LABA prescriptions (in red) and the "first" marker are filtered out. **Step 4:** Data is filtered to 6 months around the alignment point.

transpired. Users frequently need to manipulate their data, not just by removing events, but by transforming the way it's represented, a process typically known as data wrangling [45, 46].

To discover the types of data transformations that were needed beyond filtering, I spent extensive time with all of my collaborators as they explored their datasets. Based on their difficulties and feedback, I designed a series of transformation-based simplifications that met universal demands across all five research groups. These simplifications were deployed, one by one, in order to gain an initial understanding of whether they were effective at reducing complexity.

6.3.1 Interval Event Merging

In many cases, a given dataset is not only unnecessarily complex, but it is more complex than the events that actually transpired. For example, in the LABA dataset, prescription intervals were calculated based on the dispensing date and the intended duration of the prescription. The start and end date of each prescription then, was dictated substantially by the behavior of the patient: Perhaps it was convenient for the patient to refill their asthma medication a few days before the previous one ran out. In the raw event data, this presents as a series of overlapping and disjoint medication intervals.

When researchers at the PVC analyze medication use, however, they are primarily concerned with exposure. That is, the time during which the patient was actually taking a medication and exposed to its effects. Exposure can only be roughly inferred from a series of prescriptions, but it can generally be assumed that if a patient receives a series of thirty-day prescriptions, with each prescription starting a few days before or after the previous one ends, that this indicates a continuous, steady exposure. Short gaps and overlaps between successive prescriptions do not necessarily imply that the patient stopped taking the medication or took a double dose during those few days. It is important to distinguish this scenario, which results from an obvious and common behavior, from situations where

a combined dose of a single medication is a critical and interesting phenomenon.

In order to better represent exposure, and not to clutter the display with clinically uninteresting information about the exact dates of prescription refills, EventFlow provides a simple interface that allows for multiple interval events of the same category to be merged into a single interval event (see Figure 6.4). This can be done in two ways: eliminating gaps of a certain duration, and eliminating overlaps of a certain duration. This feature was designed to serve as a shortcut to two subsets of Allen's 13 interval relationships [6]: disjoint intervals and overlapping intervals. Much like Morchen's hierarchical interval model [71], interval merging focuses on duration and coincidence, rather than the exact sequence of interval end-points.

For the LABA study, any prescriptions within 14 days of each other (a standard allowance) were merged into a single exposure interval. This simplification was performed for nearly every medication category in the dataset. Interval merging can be easily undone, or saved permanently as a new dataset.

6.3.2 Category Merging

Category merging allows for multiple event categories to be combined into a single meta-category. This feature is particularly useful when individual medication types can be merged into an overarching drug class, or when drugs are prescribed in tandem. In the LABA study, three different medications were considered to be similar, low-dose treatment strategies that could serve as a valid precursor to a LABA prescription. These medications were merged into a single event category entitled "Low-Dose."

Unlike category filtering, category merging does not reduce the number of events being displayed. However, these meta-categories increase the chances that two records will have the same event sequence, and thus be merged into larger bars in the aggregated display (see Figure 6.5).

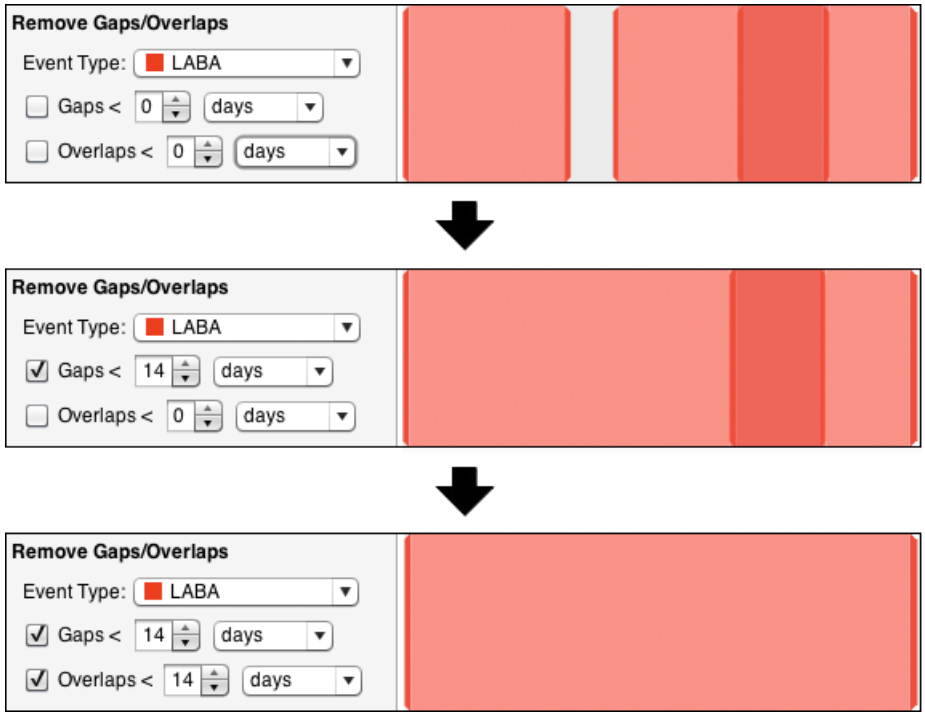


Figure 6.4: A series of disjoint and overlapping intervals are merged into a single interval.

6.3.3 Marker Event Insertion

In many datasets, certain event types repeat over and over again throughout a single record. However, for a particular question, only one of these occurrences may be of paramount interest. Marker events allow users to insert a new event at any alignment point within the data. For example, in the LABA study, the PVC researchers aligned their dataset by the LABA event flagged with the "first" attribute, and inserted a marker event at this point in each record that was (eventually) called "New LABA" (see Step 1 of Figure 6.3). For questions pertaining only to the "new" LABA prescription then, the entire category of LABA prescriptions could be filtered from the dataset, leaving only the single "New LABA" marker. Swapping an entire category for one event decreases the number of visual elements in the display and increases the chances of larger, more information dense elements.

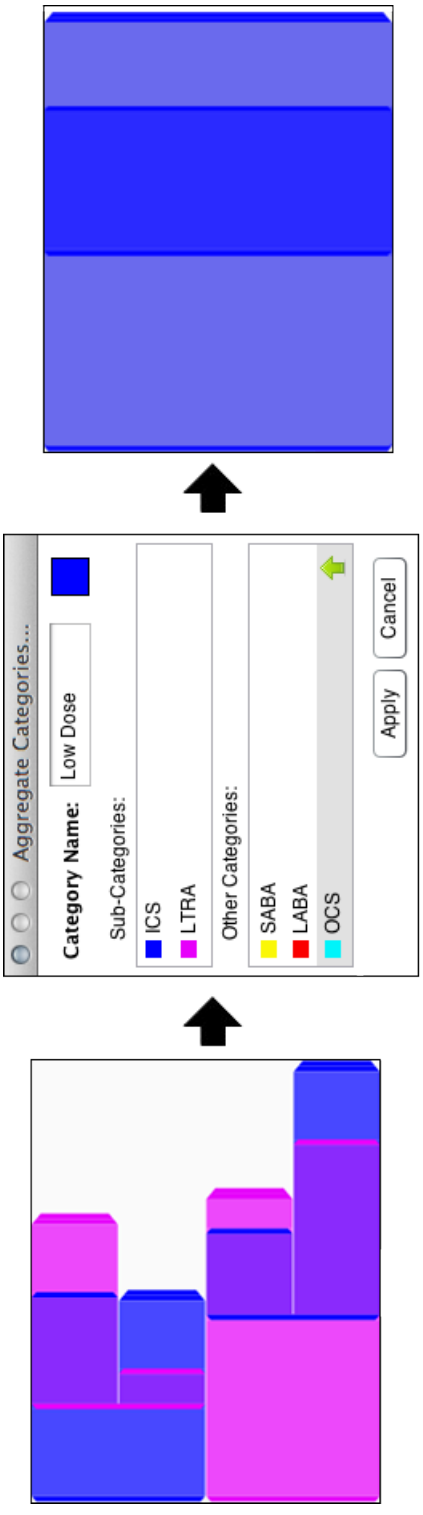


Figure 6.5: Four different event permutations (left) involving an inhaled corticosteroid (ICS - in blue) and a leukotriene-receptor antagonist (LTRA - in pink) are aggregated into a single grouping (right) when these two categories are merged together.

6.3.4 Event-Attribute Switching

As described in Section 6.2, attributes can be assigned at either the record level, or the event level. Event-level attributes can be extremely effective at reducing complexity by off-loading the event details to the attributes, which allows for fewer event categories. The basketball datasets described in Section 6.7 serve as the best example of complexity being masked in the event attributes. Instead of having event categories such as "Three-Point Shot" and "Two-Point Shot," there can just be a single category entitled "Shot." The type of shot can then be included as an event attribute. This is true of nearly every category in these datasets. For example "Rebounds" can be either "Offensive" or "Defensive," and "Timeouts" can be called by either a "Team" or an "Official."

Thus, off-loading the event details to the attributes can significantly reduce the number of event categories. However, these details must still be accessible. To account for this, EventFlow allows users to switch an event category name with the value of one of it's attributes. For example, "Shot" events become either "Three-Point Shot" events or "Two-Point Shot" events, and "Rebounds" events become either "Offensive Rebound" events or "Defensive Rebound" events. The switch is done at the category level. That is, for a switch to be completed, every event in the designated category must have the attribute in question.

Event-attribute switching allows for complexity to be reinserted only when it is needed. This is particularly useful when events have more than one attribute. For example, a "Shot" event can be either "Three-Point" or "Two-Point," but it can also be either a "Jump Shot" or "Layup." There might also be a player associated with the shot, or a type of defense that the shot was taken against. Instead of an exponential explosion in event categories to account for all of these attributes, they can simply be inserted when they are pertinent to a given question, and removed when they are not.

6.4 A Universal Transformation System

EventFlow's filter and transformation-based simplifications proved to be extremely effective in simplifying datasets, both by removing unnecessary records, and by re-representing the events in the remaining records. So effective, in fact, that their deployment was immediately followed by a barrage of requests from users for increasingly customized simplifications. In some cases, the feature being requested was specific to a single study, or even a single research question. It did not make sense to continue to implement these features on an ad hoc basis as separate interfaces within EventFlow. The result would have been an extremely cluttered and complex application. Upon further reflection, however, it became apparent that all of these simplifications had the same general objective: users wanted to find a certain event or pattern of events, and replace it with a more meaningful representation.

In addition to EventFlow's simplification and transformation features, development had just been completed on a graphic-based, advanced query system. The advanced query system, described in detail in Chapter 4, allows users to draw out their query using the same icons that are used to represent events in the individual record display. The goal of this query system was to give users more precise control over inter-record simplification. Users could select either the records that matched their query, or the records that did not match their query, and then remove the selected records from the dataset. The system had been extensively tested both in lab settings and with users to evaluate accuracy and usability, including the clarity of the query specification language and the handling of false positives/negatives. The question then, was whether this existing system could be leveraged to also perform intra-record simplification.

6.4.1 Find & Replace

Find & Replace functionality can be found in a wide range of applications including word processing and spreadsheet manipulation. It is a straightforward concept with which users in any domain are familiar. In EventFlow, the concept for a Find & Replace feature emerged naturally as increasingly more requests required this coupled functionality. Thus, in order to give users more precise and customizable control over intra-record simplification, EventFlow's advanced query system was augmented to include a replace feature. This allows users to not only find a sequence of events, but to replace it with an event sequence of their choosing.

The Find & Replace feature, which was initially developed with the help of Rongjian Lan, Hanseung Lee, and Allen Fong, can be engaged at any point in the query specification process. Similar to the Find & Replace functionality in other applications, the advanced query panel was extended to include a "Replace" button. Clicking this button brings up the Find & Replace panel where users can specify a replacement sequence, as well as continue modifying the query sequence (see Figure 6.6). The replacement sequence can be fine-tuned in five ways, listed below.

Event Specification

Events are added to the replacement sequence in the same manner that they are added to the find sequence: by clicking anywhere on the "Replace" section of the canvas. When the canvas is clicked, users are presented with the option to add either an event of an existing category, or an event of a new category (shown in Figure 6.7). For new event categories, the user must specify whether it will be a point or an interval category, and specify the category name. They can also customize the color of the new category. New event categories are added to the legend and treated as a normal category. They will appear in the existing category drop-down menus during subsequent additions to the replace sequence. That is, if

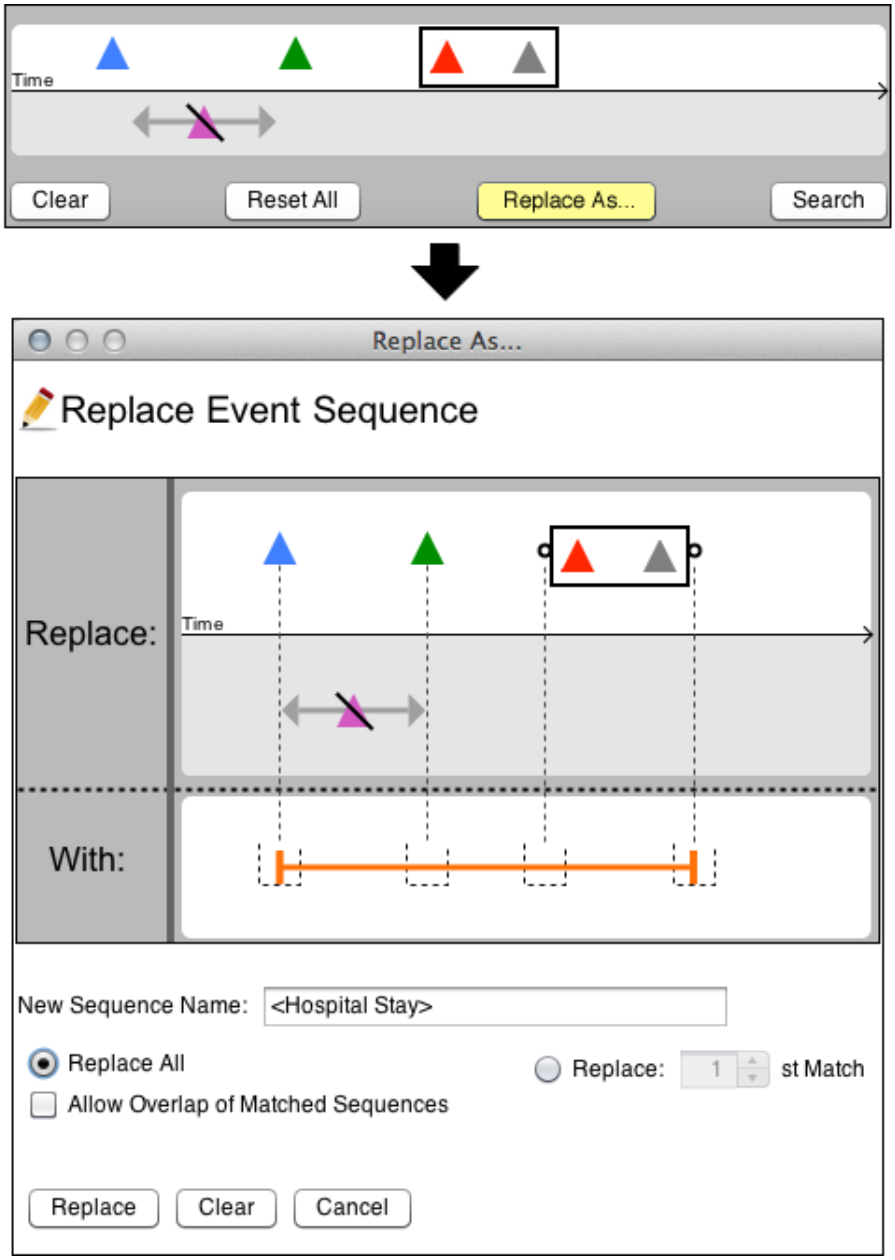


Figure 6.6: Replacements are specified by clicking the "Replace As" button on the advanced query panel. This displays the replacement panel, where users can add replacement events by clicking the "Replace" section of the canvas. Here, a series of point events is replaced by a single interval that represents the duration of the sequences (Duration Replacement)

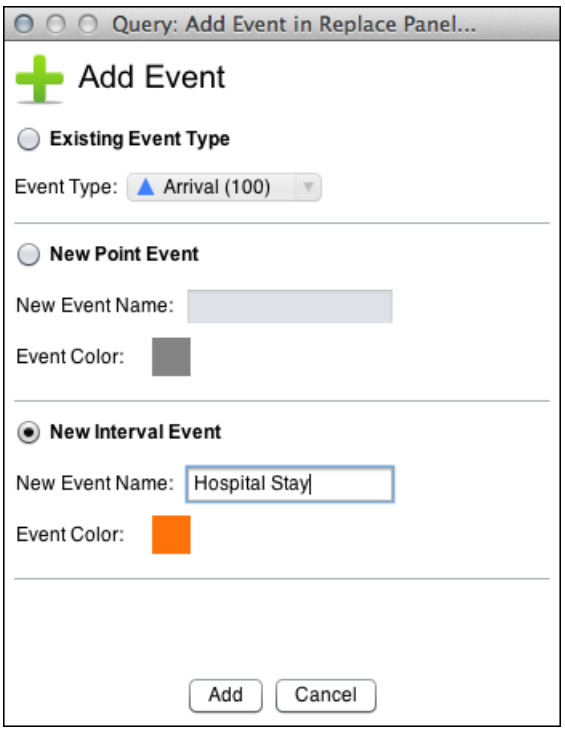


Figure 6.7: Replacement events can be specified as either an event of an existing category, or an event of a new category.

an event of a new category is to appear multiple times in the replacement sequence, it need only be created once.

Sequence Alignment

As shown in Figure 6.6, users can align the replacement sequence to the find sequence using the dotted buckets that drop down from each event in the find sequence. These are referred to as "replacement slots." Events that are added to the replacement sequence will snap to the nearest slot based on where the user clicked to add the event. Interval event end-points will snap to the nearest slots on either side of where the click occurred. Similar to the advanced search query canvas, users can drag event points in the replacement sequence to place them in the slots of their choosing. Replacement slots can be left empty, or multiple events can be added to the same slot.

For query constraints such as repetition and permutation, users are given a replacement slot at both the start and the end of the sequence (see Figure 6.6). There is no replacement slot allocated for the events inside the repetition or permutation constraints, because their exact time is less meaningful than the duration of the whole sequence. Additionally, there are no replacement slots for the absence of events because they are a semantic requirement on the search result and are not matched to a specific event or time. Metric constraints are also not yet supported in the replacement canvas. For example, users cannot search for a Stroke event and insert a replacement event exactly three days before the Stroke occurred. Replacement events can only be added to the replacement slots.

When the replacement is executed, each new event that is added will be given the exact timestamp of the event that corresponds to the replacement slot in which it was placed. In this way, new event categories can be used to highlight a wide range of temporal phenomena that are typically less evident in the visualization. This is discussed in greater detail in Section 6.4.3.

Repetition

When specifying a find sequence, it is possible for that sequence to be matched multiple times within a single event record. In this case, users have the option of replacing every instance of the find sequence, or replacing only one instance of the find sequence. If only one instance of the find sequence is to be replaced, users can choose which instance it will be. This makes it possible to replace sequences for the "first heart attack" or the "third hospital stay." This feature was designed to mimic EventFlow's alignment capability, allowing users to create alignment events based on more complex temporal phenomena.

Overlapping

The overlapping option dictates where the search engine will start looking for the next query match after a replacement is done. When a replacement sequence

is inserted into a record, the search for the next match can start either at the beginning of that replacement sequence or at the end of that replacement sequence. If replacement overlapping is allowed, the search for the next match begins at the beginning of the last replacement sequence. That is, a replacement sequence that has already been inserted into the record can be replaced again with subsequent replacement. This option is critical for replacements such as interval merging, which is demonstrated in Section 6.4.3.

6.4.2 Insert and Undo

Find & Replace can also be used to insert new events (without removing existing events), which allows users to see exactly which events are being selected by the "find" component of this feature. The system can also be used to delete events (by replacing a search sequence with an empty replace sequence). The replacement patterns are all catalogued in the Pattern tab of the control panel (see Figure 6.8), where users can revert the replacement sequence back to the original find sequence. Since replacements can potentially remove events that were created by previous replacements, the undo feature of the Pattern tab functions as a stack. A replacement can only be undone if there have been no subsequent replacements. If there have been, those replacements must be undone first.

6.4.3 Usage

Perhaps the most notable feature of Find & Replace is that it can be used to replicate the functionality of most filter and transformation-based simplifications in a single interface. As a result, the overall application can be trimmed down to a much simpler set of interfaces and controls without compromising any functionality. Alternatively, the more specialized simplification interfaces could remain in the application to serve as introductory training for the more complex, Find & Replace interface. This section provides examples of how existing features, as

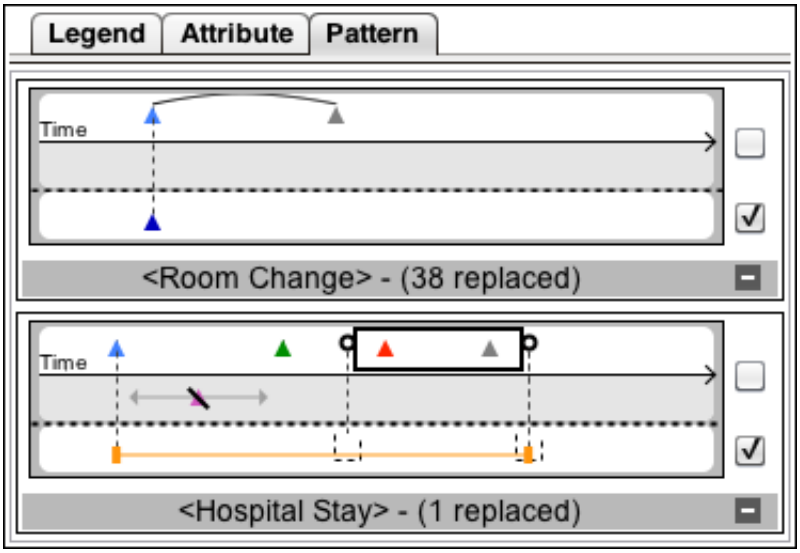


Figure 6.8: Replacements can be undone in the reverse order in which they were performed using the Pattern tab on the control panel.

well as extensive new functionality, can be accomplished using the Find & Replace interface:

- **Filter By Category** - Figure 6.9
- **Filter By Time** - Figure 6.10
- **Interval Event Merging** - Figure 6.11
- **Category Merging** - Figure 6.12
- **Marker Events** - Figure 6.13
- **Duration Replacement** - Figure 6.6
- **Events in Context** - Figure 6.24
- **Consecutive Events** - Figure 6.14

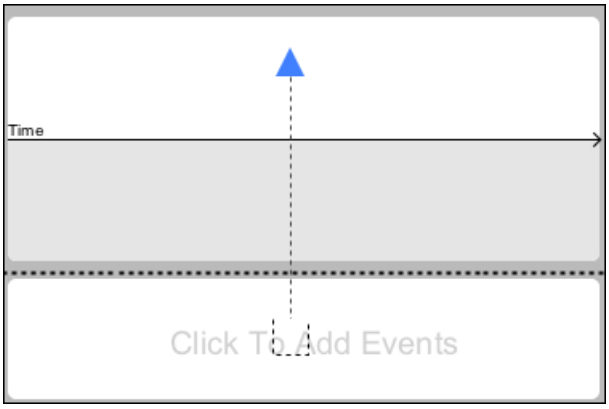


Figure 6.9: Filter By Category: Category filtering can be accomplished by replacing an event of a specific category with an empty replacement sequence. By allowing this replacement to occur multiple times, every event of the specified category will be removed.

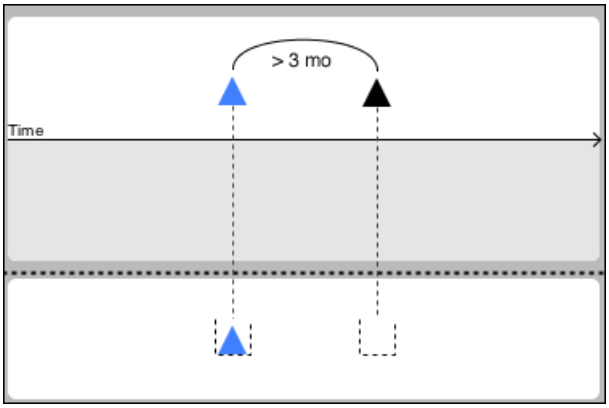


Figure 6.10: Filter By Time: Events can be filtered around an alignment event in the record using the time lapse constraint in the advanced search, as well as the wild card event. This replacement sequence filters *any* event (in black) that is more than 3 months away from the alignment event (in blue).

6.4.3.1 Query Simplification

Temporal event query is critical for isolating sets of patient records that contain a particular event pattern. One of the best strategies for executing these complex queries without making errors is to perform them incrementally, narrowing in on the population of interest step by step. In command-based query tools, this is accomplished using the AS statement to save fields and tables as new entities. However, the majority of these command-based tools are designed for relational

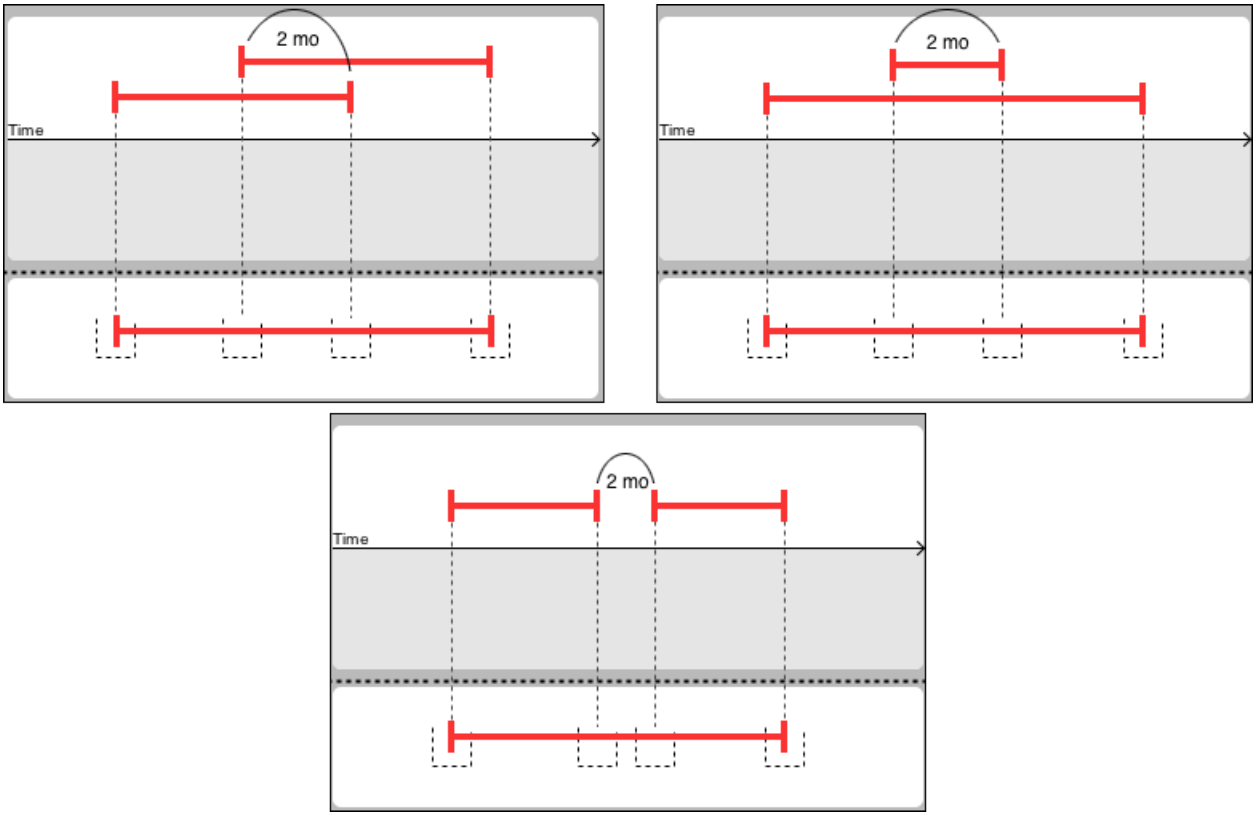


Figure 6.11: Interval Event Merging: Interval event merging can be accomplished using three separate replacements. One replacement eliminates the gaps between events (bottom), and the other two eliminate overlaps (top). The duration of the gap or overlap can be specified using the time lapse constraint.

data structures, and do not provide direct support for temporal event sequences.

Find & Replace allows users to take a similar incremental approach to executing complex queries. Separable components of the query can be executed independently and flagged with Find & Replace-generated marker events. These marker events can then be incorporated into subsequent queries that dictate how the marker events must relate to each other. For example, consider a study that is focusing on patients who were treated with Medication X for at least 3 months, and then experienced Symptom Y within one month of stopping X. However, maybe Medication X is prescribed in month-long intervals, requiring researchers to manually construct the actual duration of exposure by piecing together each

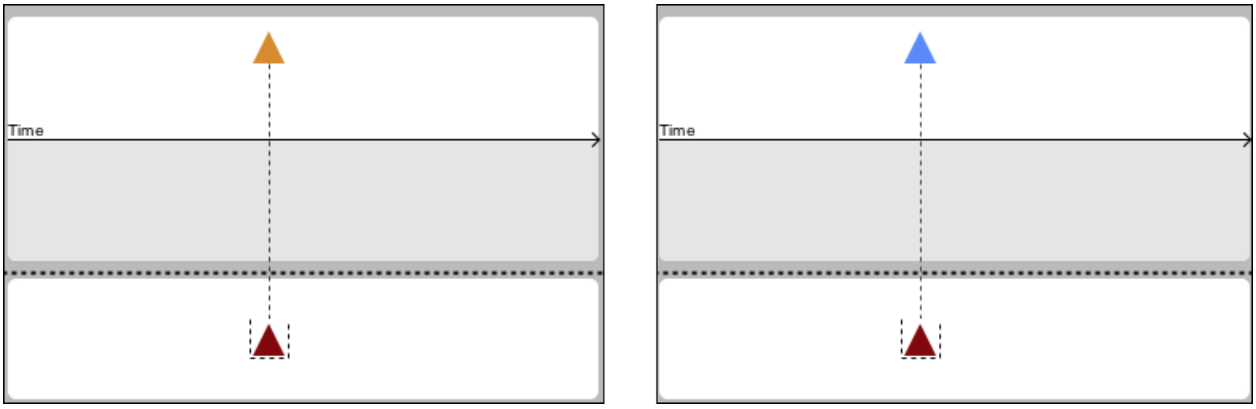


Figure 6.12: Category Merging: Similar to filtering by category, category merging is done by replacing every event for multiple categories with the same event of a new category. Here, two existing categories (in orange and blue) are replaced by a new category (in red).

prescription (interval merging). Maybe Symptom Y is not a single event, but two different events that can occur in any order, so long as they occur within a week of each other.

Identifying Symptom Y could begin with a query that replaces any permutation of these two events with a marker interval that spans the duration between them. Records containing a marker interval with duration of less than a week could be queried for and isolated. Similarly, merged Medication X intervals that exceeded 3 months could be replaced with a second marker interval. These two artificially generated marker events could then be combined to formulate the final query for qualifying patients. This process, in its entirety, is shown in Figure 6.15.

6.4.4 The LABA Simplification

In the LABA study, Find & Replace was used when the researchers observed that LABAs are typically prescribed in tandem with an Inhaled Cortical Steroid (ICS). The result, in the aggregated display, is a sequence of four event points that essentially represent one treatment. This pattern was simplified by replacing it with a single interval, representing the combined prescription of LABA + ICS

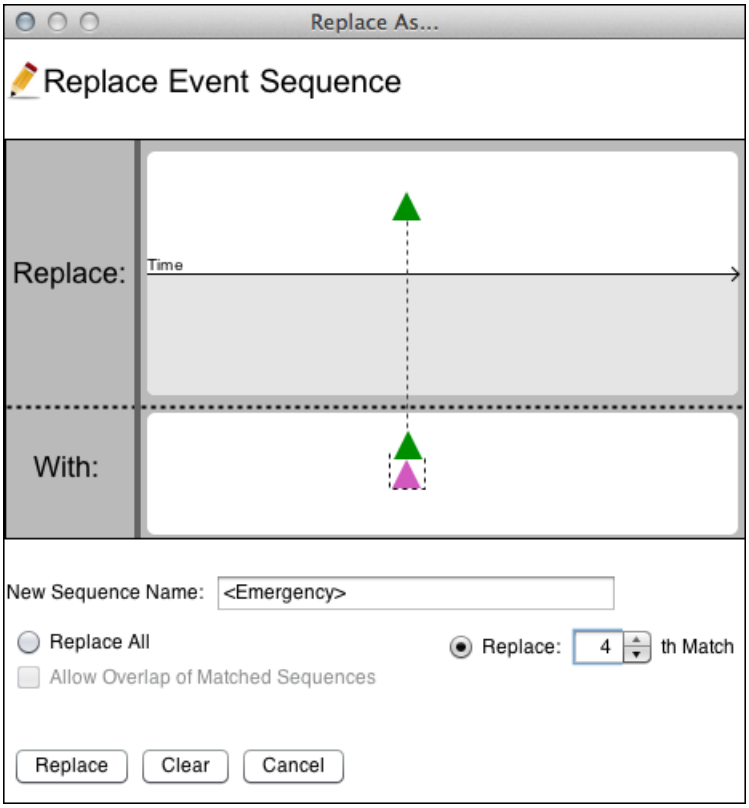


Figure 6.13: Marker Events: Marker events are inserted by identifying the insertion point in the same way that alignment points are identified. Here, a marker event (in pink) is inserted at the 4th green event.

(see Figure 6.16). Furthermore, as described previously, each record contained a point event that marked the start of a "new" LABA prescription. This marker and the interval it tagged were replaced by representing it as simply a "New LABA" interval. In total, the following sequence of simplifications were performed on the LABA dataset:

1. Add marker event for "new" LABA prescriptions.
   2724 elements, 1.14% of display height per element

2. Filter out extraneous categories.
   2694 elements, 1.15% of display height per element

3. Replace concurrent LABA and ICS prescription with single LABA + ICS

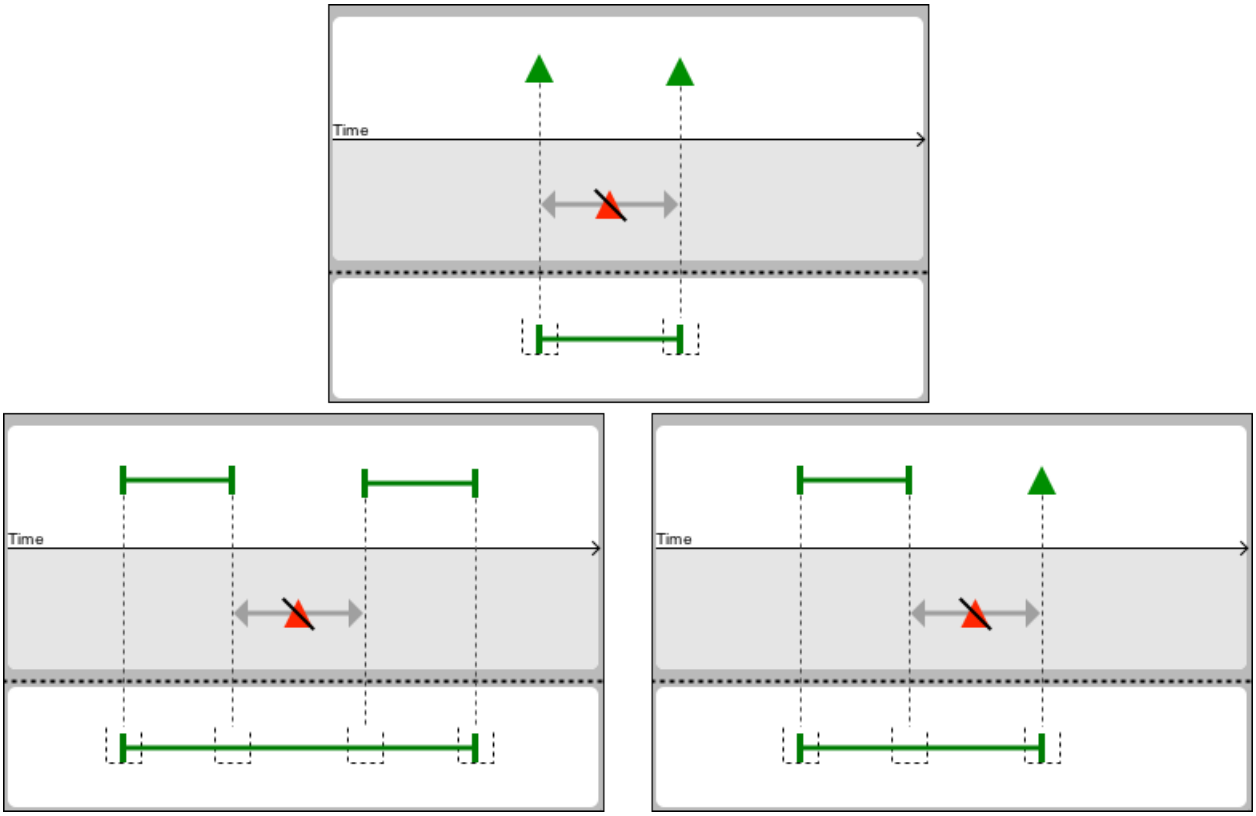


Figure 6.14: Consecutive Events: Here, an uninterrupted sequence of point events (in green) is replaced by a representative interval. This is done in three steps. First, consecutive green point events are replaced with an interval (top). Then, consecutive green intervals are merged, as well as a green interval followed by a green point event.

interval.

2061 elements, 1.16% of display height per element

4. Replace LABA prescription marked as "new" with a single New LABA interval.

   1993 elements, 1.15% of display height per element

5. Merge intervals to reflect exposure.

   1450 elements, 1.19% of display height per element

6. Aggregate similar medications into single category.

   1332 elements, 1.25% of display height per element

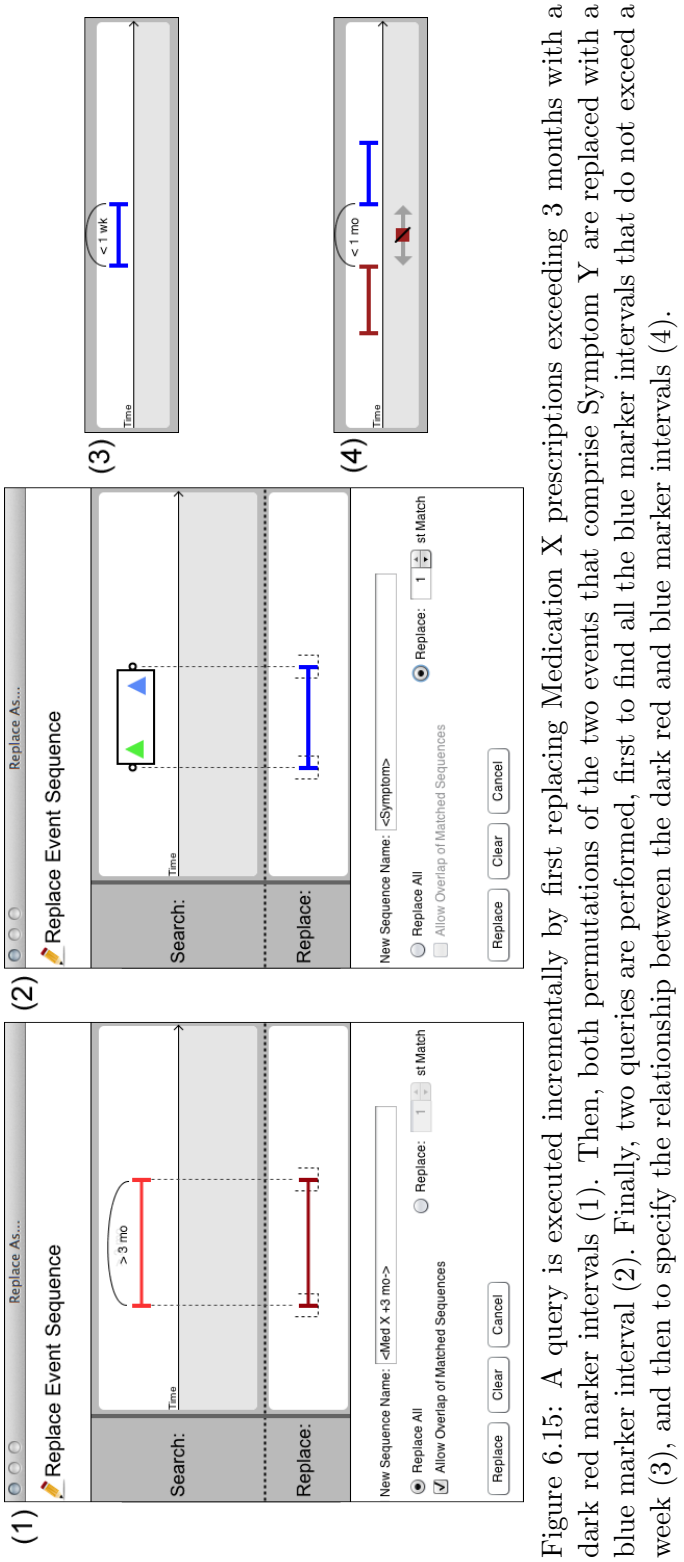


Figure 6.15: A query is executed incrementally by first replacing Medication X prescriptions exceeding 3 months with a dark red marker intervals (1). Then, both permutations of the two events that comprise Symptom Y are replaced with a blue marker interval (2). Finally, two queries are performed, first to find all the blue marker intervals that do not exceed a week (3), and then to specify the relationship between the dark red and blue marker intervals (4).

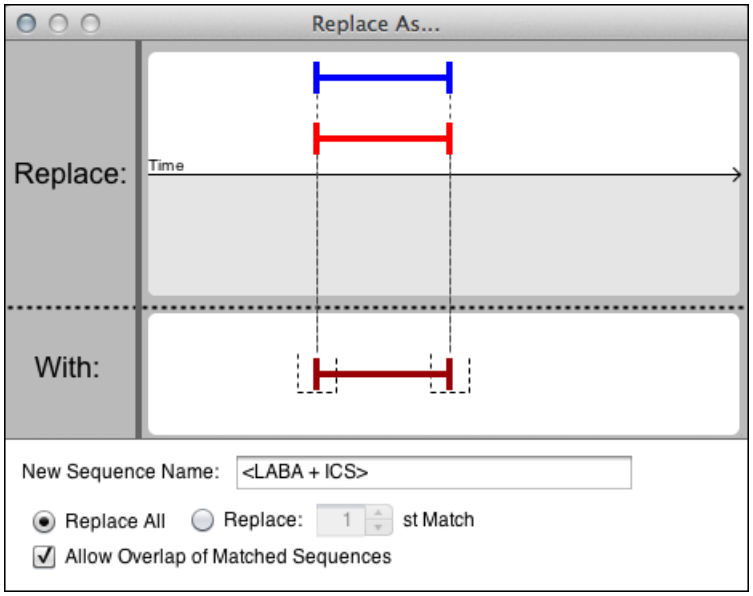


Figure 6.16: Concurrent LABA (in bright red) and ICS (in blue) prescriptions are replaced by a single interval representing the combined treatment of LABA + ICS (in dark red).

7. Align by point of interest.

   1123 elements, 1.56% of display height per element

8. Filter display to the six months on either side of the alignment.

   491 elements, 2.08% of display height per element

The resulting, aggregated display is shown in Figure 6.1-right. This simplified display consists of only 492 visual elements, each averaging 2.08% of the display height per element. This constitutes an 81% reduction in visual elements, and an 82% increase in the average size of each element. It becomes possible to visually extract meaningful information from the display. For example, the PVC researchers immediately observed that a significant number of patients had not received any other treatment in the six months prior to their LABA prescription, which is not the recommended practice. This was a new and valuable insight that generated countless other questions.

It is critical to point out that this series of simplifications to the LABA dataset was only an *initial* pass to narrow the data down to the general theme of the research objective. As questions get more specific, the dataset can be further simplified. Modified records can be saved as a new dataset, allowing users to generate, manipulate, and share data that more accurately reflects the events that transpired and the overarching research objective.

6.5 Opioid Misclassification

In addition to the LABA study, the researchers at the PVC were investigating opioid use as part of a second long-term study. Opioids, a class of medications that are typically prescribed for pain management, can be habit-forming when taken improperly, and are blamed for an increasing number of drug-related deaths [12]. In the most recent phase of this work, researchers were trying to determine within a sample dataset of 1000 records if patients that had been previously classified as acute users were actually misclassified chronic users. These classifications are determined by reviewing a series of prescriptions, which could be for either high or low dose opioids, and consolidating them into "episodes" that are then classified as acute, intermediate or chronic. The question was whether patients with repeated acute episodes should be classified as chronic users.

The first step of the simplification process was for the researchers to take the raw prescription data (1064 visual elements, with each element averaging only .49% of the total display height), and restructure it into acute, intermediate, and chronic episodes. An episode was defined as any consecutive prescriptions within 14 days of each other. To account for this, they first merged the high and low dose prescriptions into a single event category, and then used the Find & Replace system to mark the first prescription in each episode (see Figure 6.17).

From here, they replaced episodes with acute, intermediate, and chronic events based on the duration of the episode and the number of prescriptions. For ex-

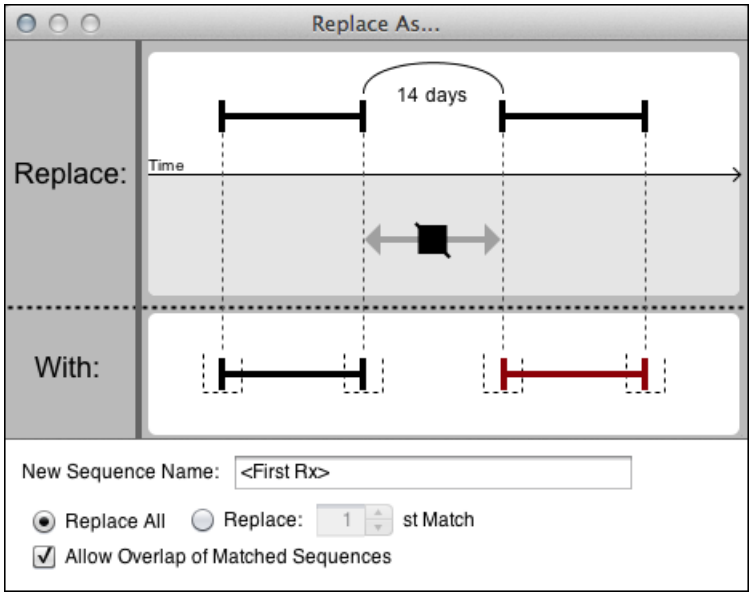


Figure 6.17: The first prescription in each episode, defined as any prescription preceded by 14 days without another prescription, is replaced with a marker interval (in red). A second replacement was also done to replace the first prescription of each record, which would also be considered the start of an episode, with the same marker interval.

ample, acute episodes were any episodes under 60 days and consisting of two or fewer prescriptions. These were identified by first replacing one and two prescription intervals with a placeholder interval. Then, placeholder intervals lasting less than 60 days were replaced by a new event representing an acute interval. Similar strategies were use to identify chronic episodes, and any remaining episodes were replaced as intermediate episodes. The resulting dataset consisted of only 180 visual elements, each averaging 1.96% of the total display height, an 83% reduction in visual elements and almost a 300% increase in the average element height (see Figure 6.18 - Steps 1 $\rightarrow$ 2).

Step 1: 1064 elements, .49% of display height per element

Step 2: 180 elements, 1.96% of display height per element

Using this simplified dataset, it was immediately apparent from the aggregated

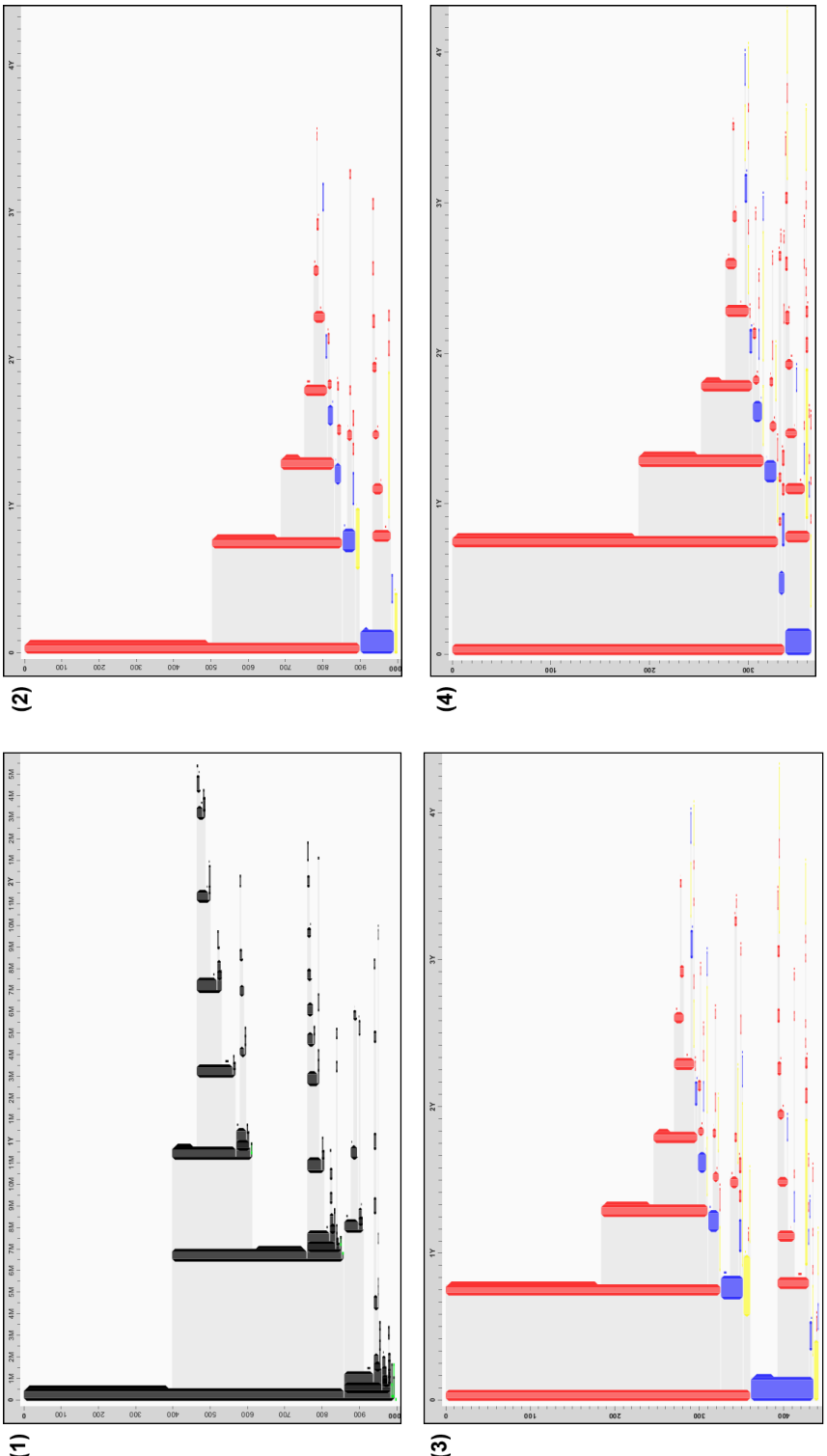


Figure 6.18: Were chronic opioid users misclassified as acute users? - **Step 1:** The original Opioid dataset consists only of raw prescription data. **Step 2:** The dataset is restructured into acute (red), intermediate (in blue), and chronic episodes (in red). **Step 3:** Patients that only had one acute episode are removed. **Step 4:** Patients without consecutive acute episodes are removed.

display that the majority of acute users had been classified correctly. These patients had only one, acute episode, and were easily distinguishable from the rest of the population that consisted of more varied episodes. Using selection, these patients were removed from the dataset, reducing the number of patients by over 60% in a single click (see Figure 6.18 - Steps 3). This filtering did not significantly reduce the number of visual elements, but increased the average height of the remaining elements to 3.04% of the total display height.

Step 3: 180 elements, 3.04% of display height per element

The researchers then narrowed the dataset down further, using query, to include only the patients who had consecutive acute episodes. From here, they could scroll over the time lapse between each pair of acute episodes to see the distribution of when each patient started their second episode in relation to ending their first. If these patients were exhibiting chronic behavior, the distribution would have been heavily front-weighted. That is, patients would be starting their second acute episode very soon after their first.

Step 4: 156 elements, 3.79% of display height per element

However, this was not the case. The lapse of time between when patients started their second episode, relative to the end of their first episode, was uniformly distributed. This indicated that there was no obvious reason to believe that a significant portion of these patients had been misclassified. The final consensus was that patients had been accurately classified as either acute or chronic users. Figure 6.18 summarizes the opioid simplifications.

6.6 Pediatric Trauma Procedures

Children's National Medical Center is the premier provider of pediatric care in Washington, DC, treating more than 360,000 patients each year. Throughout the hospital, doctors and medical researchers are dedicated to improving patient care and process flows. In the pediatric trauma bay, the Advanced Trauma Life Support (ATLS) protocol [104] provides a common framework for the initial assessment of injured patients. Researchers overseeing this department were interested in whether the trauma team was following this protocol, and what factors tended to result in deviations from this protocol.

The goal of the ATLS protocol is to identify and treat the greatest life threat first. Previous work has shown that adherence to this protocol can have a positive impact on patient outcomes [4, 110]. The protocol is comprised of two, sequential parts: the primary survey and the secondary survey. The primary survey is further comprised of the following checks that must be completed, which are referred to as the ABCDE's:

- Airway
- Breathing
- Circulation
- Disability
- Exposure

Once these checks have been completed, the secondary survey consists of a head-to-toe examination of the patient to assess external injuries. It is worth noting that the secondary survey cannot physically be completed unless the exposure check has been completed, so the completion of this check is typically implied by the start of the secondary survey. With that allowance accounted for, the trauma team is

expected to perform these five steps, in exactly this order, during every trauma resuscitation.

To this end, researchers collected data from 215 trauma resuscitations over a 4 month period in the pediatric trauma bay. The question of ATLS adherence was only one component of a much larger study on trauma procedures and, as a result, the dataset consisted of nearly 30 different event types (see Step 1 of Figure 6.20). The first step of evaluating this dataset was to filter out the event categories that were not relevant to the ATLS protocol.

Next, while the ATLS protocol requires only one confirmation of a pulse (Circulation), in practice, trauma nurses typically take two pulse readings from the patient: one from the patient's heart (central pulse) and one from an extremity (distal pulse). To better align the dataset with the ATLS protocol, these two pulse readings, which appeared in the dataset as separate event categories, were merged into a single category. Then, a Find & Replace was performed on patients with multiple pulse events to include only their first pulse reading (see Figure 6.19). This allowed researchers to better understand the time at which the patient's pulse was first confirmed. This process of narrowing down the dataset to best represent the ATLS protocol is shown in Figure 6.20.

Step 1: 1209 elements, .84% of display height per element
Step 2: 155 elements, 3.93% of display height per element
Step 3: 132 elements, 4.62% of display height per element
Step 4: 60 elements, 11.58% of display height per element

The resulting visualization clearly showed that the ATLS protocol had been exactly followed in approximately half of the trauma resuscitations. Of the remaining resuscitations, the most common deviation from the protocol was that the disability assessment was performed after the secondary survey had begun. However, other mis-sequencings occurred in nearly every permutation. The re-

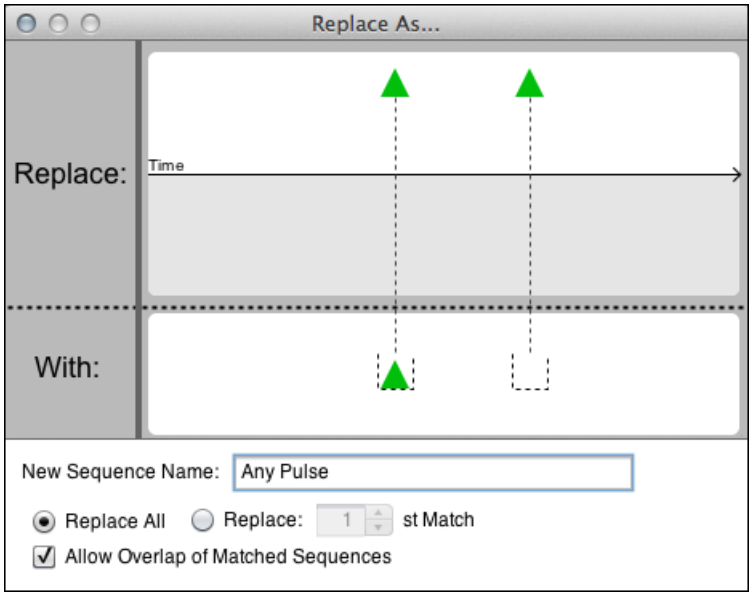


Figure 6.19: Records with multiple pulse events are replaced with a single pulse event to represent that the patient's pulse was confirmed in some form.

searchers hypothesized that the most egregious deviations from the protocol likely occurred during what they call "now" resuscitations. In these resuscitations, the trauma team is given virtually no prior notice before the patient's arrival (they are typically given about 20 minutes of notice). The team must prepare and organize themselves as the patient is being brought into the trauma bay. The researchers thought that this would contribute substantially to the ATLS protocol being performed incorrectly.

To investigate this hypothesis, researchers split the dataset by the "now" indicator (see Figure 6.21). To their surprise, the lack of preparation time appeared to have no effect on adherence to the ATLS protocol. Approximately the same proportion of resuscitations deviated from the protocol among "now" patients and patients for whom arrival notice was given ahead of time. In fact, the researchers were unable to identify any external attributes that significantly affected adherence to the protocol. Their conclusion then, was that deviations from the protocol were due to individual circumstances, data collection procedures, or insufficient

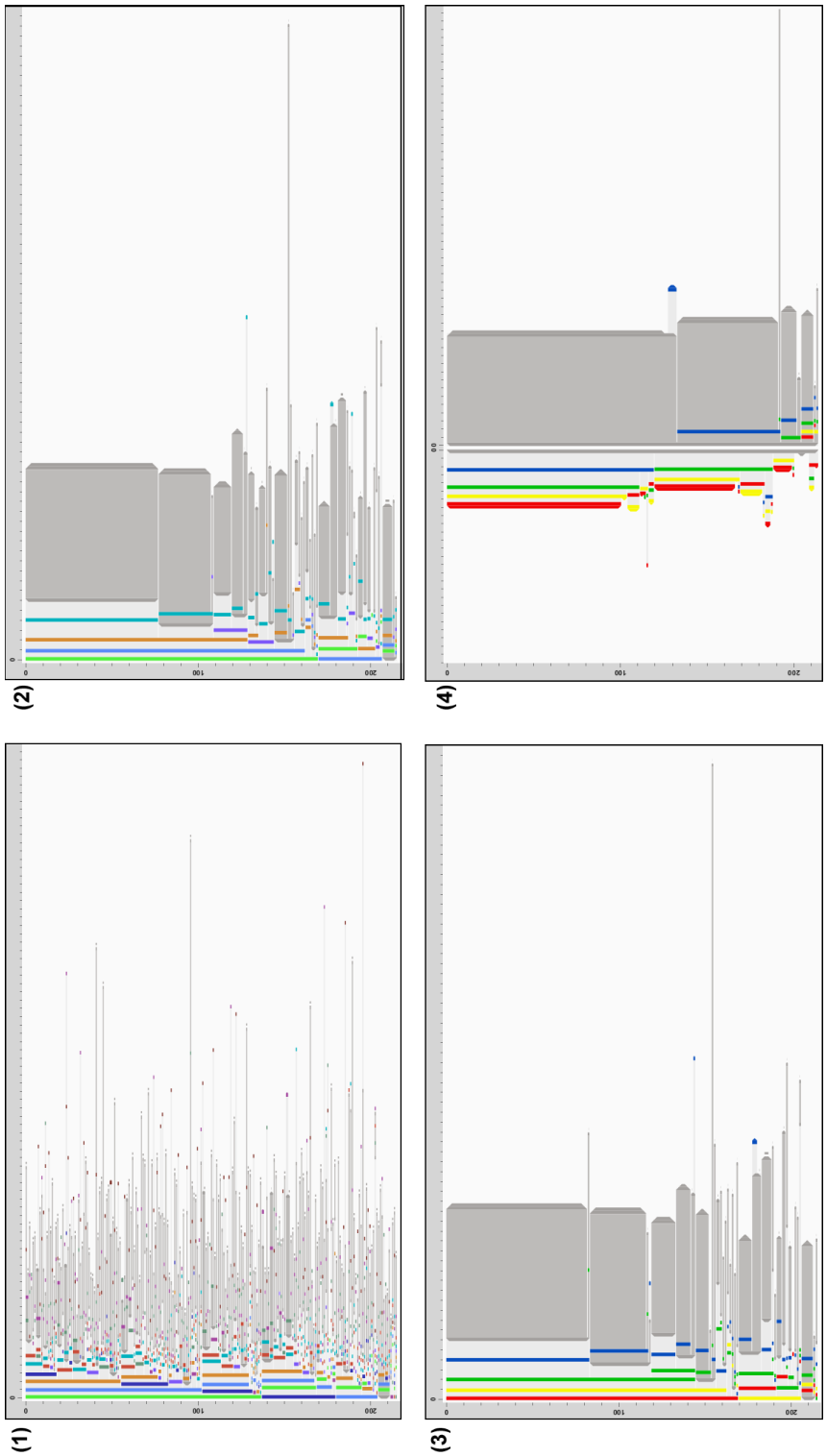


Figure 6.20: How closely were trauma procedures followed? - **Step 1:** The initial dataset is loaded into EventFlow. **Step 2:** Non-relevant event categories are filtered out. **Step 3:** The two pulse event categories are merged into one category, and an intuitive color scheme is applied. **Step 4:** Duplicate pulse events are removed and the dataset is aligned by the start of the secondary survey.

training.

Overall, the researchers were able to better understand the type and frequency of the most common deviations from the ATLS protocol. Their plan is to use this information in the design of a larger scale study in which they will try to better correlate specific mis-sequencings with patient outcomes. Their use of EventFlow's simplification features demonstrated how easily a dataset can be transformed from the representation dictated by the collection process, to the representation necessary for the evaluation process.

6.7 Basketball Play-By-Play Breakdown

To demonstrate that EventFlow's simplification capabilities can be applied to datasets outside of the medical domain, I created a dataset (or, more accurately, two datasets) based on the play-by-play statistics from the University of Maryland men's basketball team. Play-by-play data consists of the events that took place over the course of a game (shots, rebounds, steals), timestamped against the game clock. The play-by-play breakdown of every college and professional basketball game is freely available online. For this study, I looked at the March 16th, 2013 game against the University of North Carolina (UNC), a game that Maryland ultimately lost by three points. The goal of this experimental analysis was to try to determine what went wrong.

A basketball game can be thought of as a series of disjoint, alternating possessions between the two teams (see Figure 6.22). The two phenomena that a sports analyst can look at, using EventFlow, are how events affect each other within possessions, and how events affect each other across possessions. The latter of these objectives comes in the form of two questions:

- How well does Maryland transition from offense to defense?
- How well does Maryland transition from defense to offense?

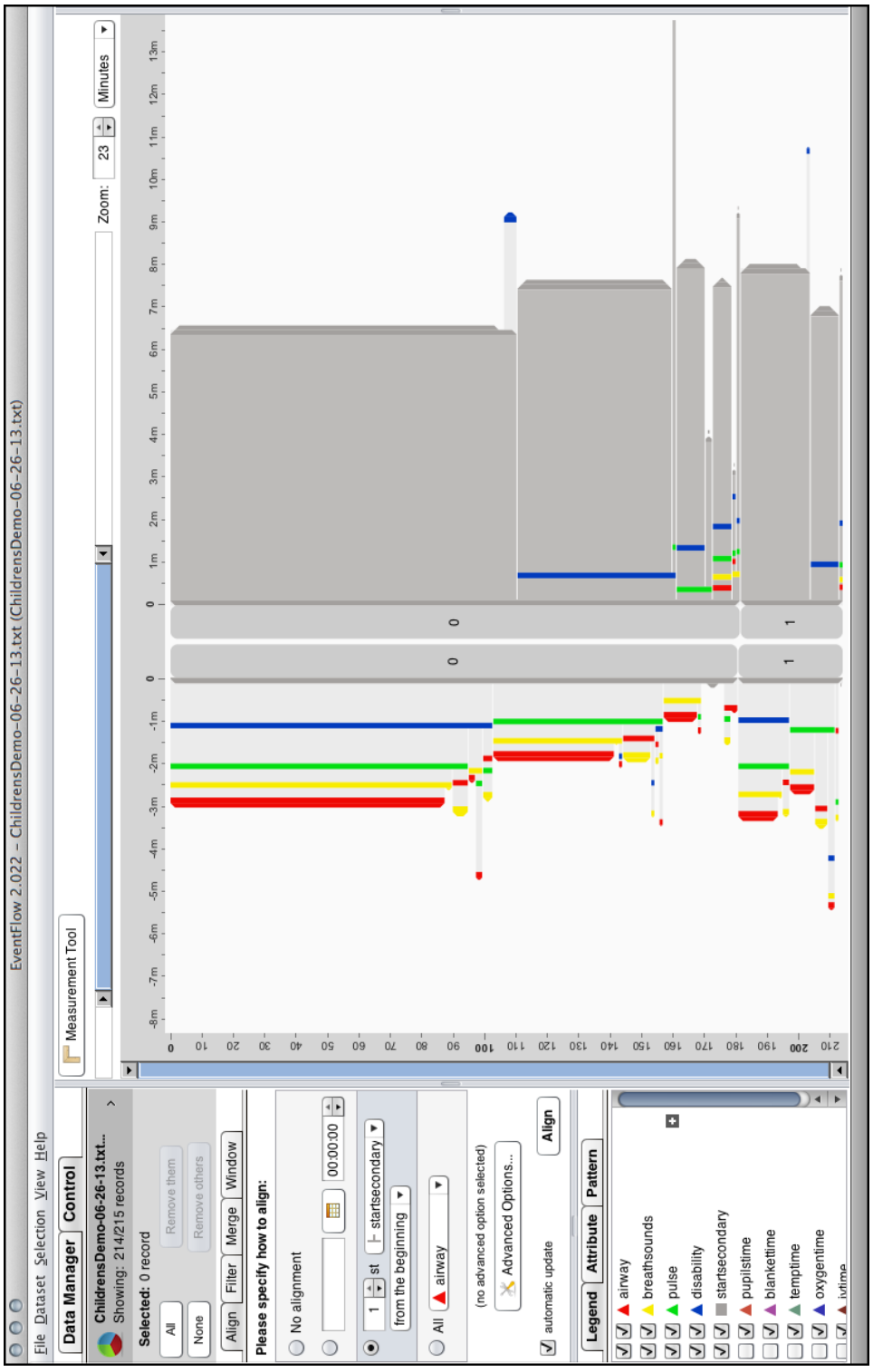


Figure 6.21: The pediatric trauma dataset is split into cases where the patient arrived with approximately 20 minutes of prior notice (top), and cases where the patient arrived without notice (bottom). There were no noticeable differences in the proportion of mis-sequenced records in these two groups.

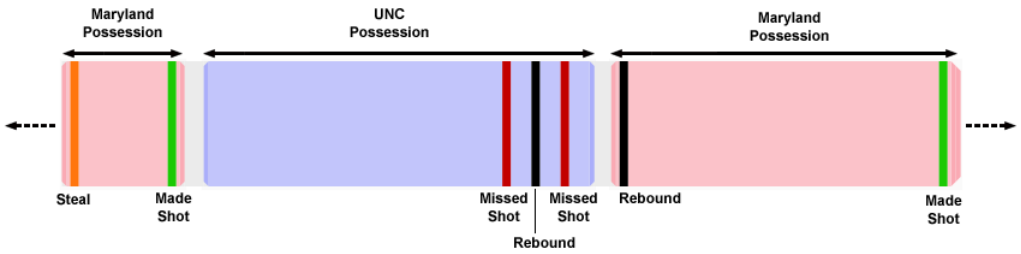


Figure 6.22: Moving through time, left to right, the game can viewed as a series of alternating possessions (Maryland in red, UNC in blue.

These questions were addressed by splitting the play-by-play data into two overlapping datasets. The first dataset takes the entire game of possession changes, and segments it into two-possession increments of a Maryland possession followed by a UNC possession. The second dataset does the same thing, but in increments of a UNC possession followed by a Maryland possession. In both datasets, each two-possession increment is treated as a record. Thus, the three-possession sequence shown above would be segmented such that the first and second possessions constituted one record in the Offense→Defense dataset, and the second and third possessions constituted one record in the Defense→Offense dataset.

From a simplification standpoint, the primary challenge of the play-by-play data was the granularity of the event categories. Nearly every category had subcategories that could be useful for certain questions, but irrelevant for others. Shots could be 3-point jumpers, 2-point jumpers, or layups. Rebounds could be offensive or defensive. Timeouts could be called by either team or the officials. Without some degree of category merging, there would be too many categories, resulting in too many colors in the display for users to visually discern between.

The strategy in analyzing this dataset then, was to begin by merging the event categories as much as possible, to produce the least visually complex display. The Offense→Defense dataset, loaded into EventFlow's aggregated display and aligned by the possession change, is shown in Figure 6.23. At this level of aggregation, it is easy to pick out the notable dynamics of the game, such as the events that

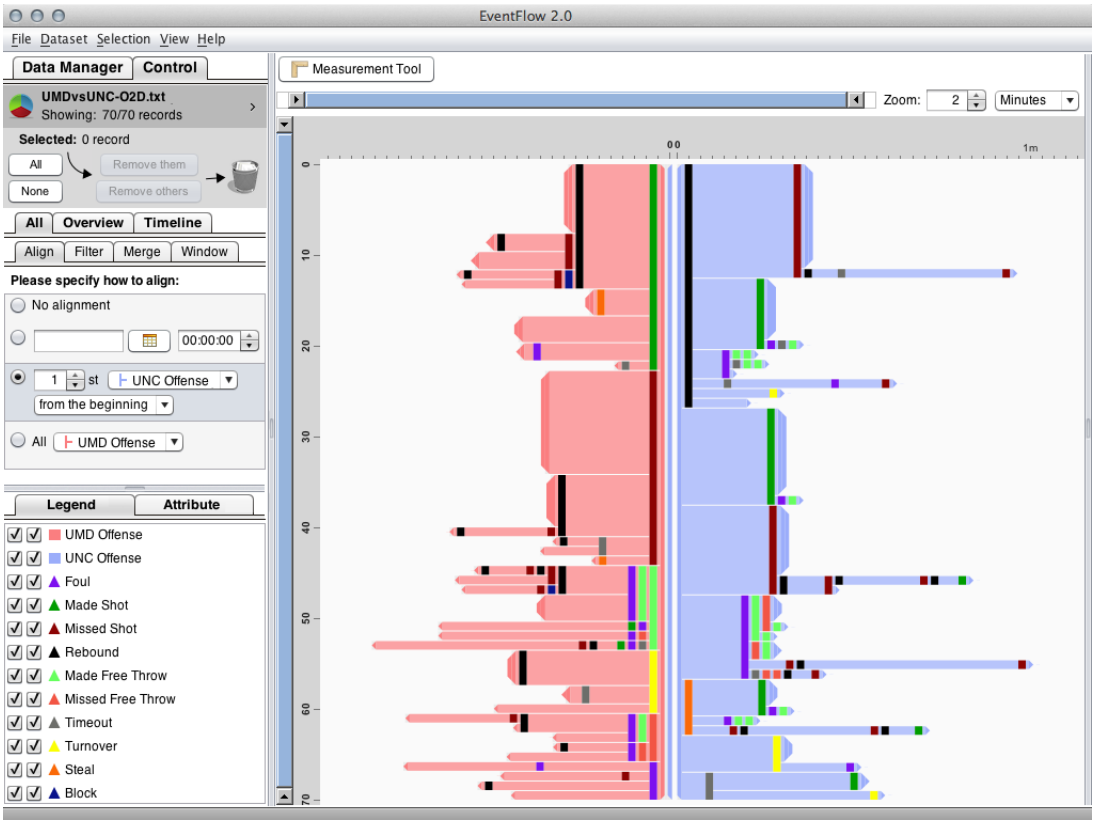


Figure 6.23: The initial aggregated display of the Offense→Defense dataset. The data is aligned by the possession change, with the Maryland possession to the left of the alignment and the UNC possession to the right.

caused the possession change, and the various scoring attempts. Essentially, the initial simplification strategy was to oversimplify.

From there, complexity could be reinserted only when it was needed for a particular question. This could be done by simply un-merging the meta-category. However, it frequently proved more effective to reinsert complexity using the Find & Replace interface, as it offered more fine-tuned control over how much complexity was being added. For example, offensive and defensive rebounds were merged into the meta-category, "Rebound." One of the questions asked of this dataset was, "Did UNC capitalize on their offensive rebounds?" This question could be answered by un-merging all of the rebound events across both possessions. However, since a

defensive rebound results in an immediate possession change, Find & Replace could be used to isolate only the rebounds in which UNC had possession and maintained possession, and replace them as "Offensive Rebound" (see Figure 6.24).

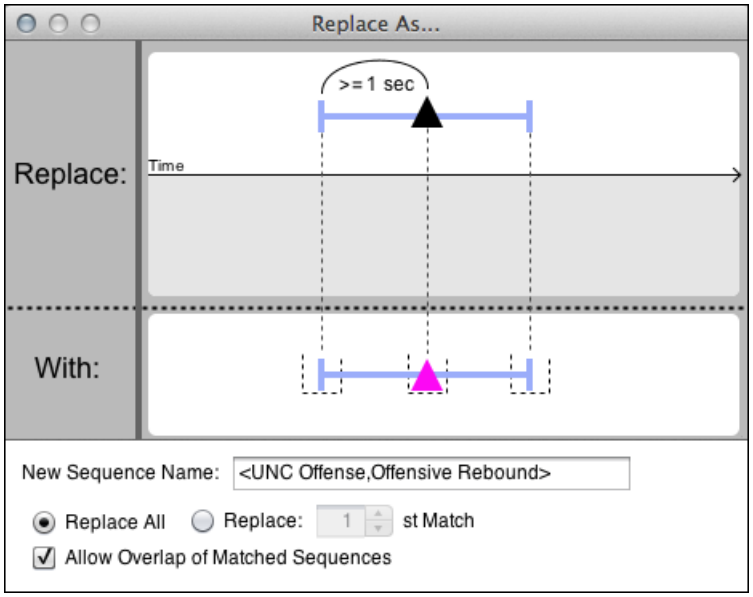


Figure 6.24: Events in Context: Rebounds (in black) in which UNC had possession and maintained possession are replaced as "Offensive Rebounds" (in pink).

The big question remained then: "What could Maryland have done in order to win?" It's an interesting question, given that the final score differential was only one basket. In fact, using various different filtering, merging and un-merging, and Find & Replace, the two datasets produced almost identical aggregations of how each team performed. However, there was one glaring difference.

An offensive possession can be thought of as starting either actively (via a steal or a rebound) or passively (via an opponent's score or a dead ball). In both datasets, the passive possession changes are isolated by removing the possessions that began with either a steal or a rebound. The remaining possessions can be further simplified by removing dead ball events, such as timeouts or jump balls. Finally, since the datasets are already filtered by the type of possession change, the preceding possession can be removed entirely (using Find & Replace). This

process, being done to the Offense→Defense dataset, is shown in Figure 6.25.

Step 1: 183 elements, 5.32% of display height per element
Step 2: 106 elements, 9.26% of display height per element
Step 3: 92 elements, 10.46% of display height per element
Step 4: 39 elements, 9.98% of display height per element

Figure 6.26 shows both of the resulting simplifications. Both teams had roughly the same number of offensive possessions coming out of passive possession changes (Maryland - 38, UNC - 37). However, UNC was clearly executing their offense more efficiently. Their most common outcome was a score which, on average, occurred much faster than the first event in any of the Maryland possessions. In passive possession change scenarios, UNC scored on twice as many possessions as Maryland did. It would be reasonable then, to conjecture that the game was ultimately decided by UNC's ability to quickly execute their half court sets. Defensively, Maryland could have improved their chances by scouting these plays, and practicing to defend them at speed.

6.8 Automated Transformation and Simplification

EventFlow not only allows a dataset to be simplified down to its most critical elements, but it also allows for those simplifications to be saved, and then applied to a new dataset. These saved simplifications are applied to the new dataset automatically as they are being loaded. This feature impacts both the scalability of the application, as well as the time required to arrive at an insight. These benefits are described below, in two representative examples.

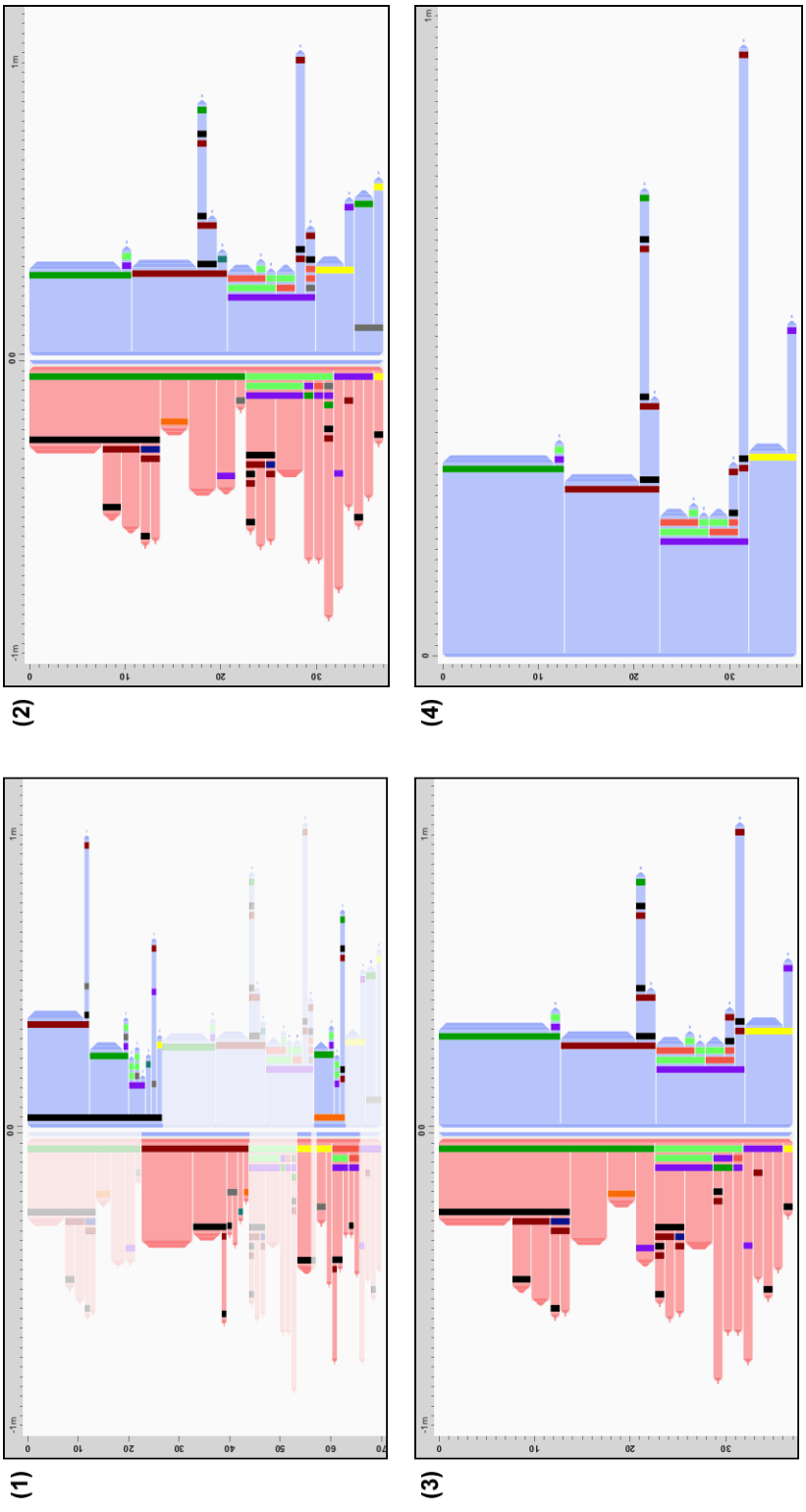


Figure 6.25: How did the UNC offense perform off of passive transitions? - **Step 1:** Active possession changes are selected from the original Offense→Defense dataset based on rebounds (in black) and steals (in orange). **Step 2:** These possessions are removed. **Step 3:** Dead ball events are filtered out. **Step 4:** The Maryland possession is removed.

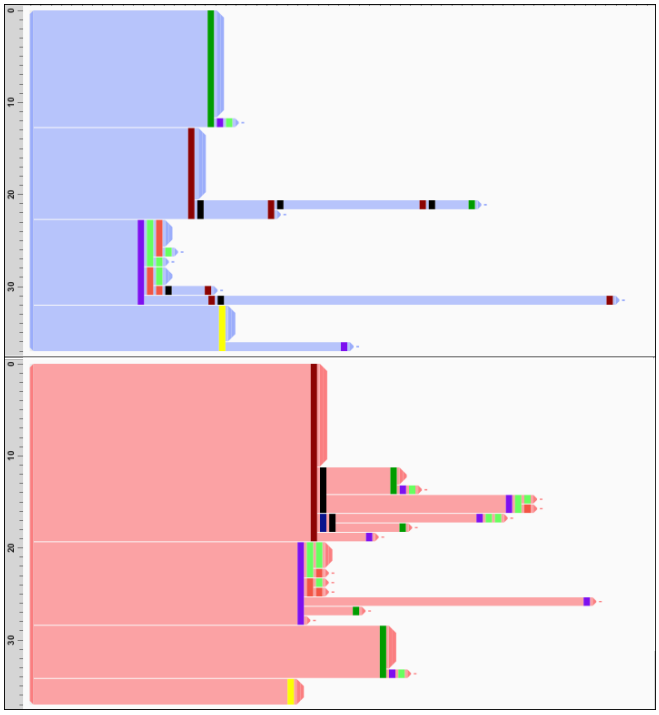


Figure 6.26: The UNC offense, coming off of passive possession changes (top) vs. the Maryland offense, coming off of passive possession changes (bottom). UNC executed their offense visibly faster than Maryland (the horizontal axis is time), converting most of these possession into points (in green).

6.8.1 Simplification for Memory

In any data analytics or visualization tool, there will always be limits to the amount of data that can be loaded and displayed. Simplification offers an opportunity to stretch these limits beyond their initial capacity. For example, researchers at the University of Florida College of Medicine were investigating symptoms of and treatments for a condition call Diabetic Foot. Their initial dataset consisted of 100 patient records, and 5,674 events. The patient records were so long and heterogeneous that they resulted in very little aggregation, and thus required too much memory to be loaded into EventFlow's aggregated display structure. However, only a subset of events from each record was really needed to answer the questions at the core of this study.

To account for this, the researchers loaded a randomly selected subset of records from the original dataset, and performed a series of simplifications to pare them down to the elements of interest. These simplifications were then saved as an EventFlow "mod" file. This mod file was then loaded in combination with the original dataset. The result was a dataset consisting of the original 100 records, but only 339 events. The automatically simplified dataset could easily be loaded into EventFlow's aggregated structure without impacting its memory capacity, and the researchers were able to continue their study on the complete set of 100 records.

6.8.2 Transformation and Simplification for Time-Saving

In many domains, it is necessary to perform the same analysis on multiple similar datasets. For example, an analyst for the Chicago Bulls was interested in how the offensive rebounding of the Indiana Pacers contributed to their offense. Again, the answer to this question lay at the end of a series of simplifications and data manipulations. Once the answer had been determined, the logical follow-up question was "How about the Miami Heat?"

In this case, the series of simplifications done to the Indiana Pacers dataset could be saved as a mod file. This mod file could then be loaded with a dataset for any other team in the NBA. The result was that analysts could save time when they had already worked through the process of making the necessary simplifications. Automatic simplification makes it easy to perform the same analysis on multiple datasets without having to repeat the same series of repetitive, and potentially error-prone interactions.

6.9 A Process Model of Transformation

Even with EventFlow's powerful data transformation capabilities, the initial display of a dataset can be overwhelming to new users, making it difficult to determine which steps should be taken first. In this section, I lay out a stepwise process for

making a dataset usable for analysis. These steps are designed to be domain and task agnostic.

Step 0: Define Task

Users typically have one objective in each of the following two categories. Identifying these objectives upfront can help users to clarify the operations that must be performed in subsequent steps.

Record level objective:

- Analyzing a single group of records.
- Comparing multiple groups of records.

Event level objective:

- Evaluating a particular event pattern.
- Evaluating a particular event interaction.

Step 1: Extract Dataset

In many cases, data must be extracted from a larger database and formatted according to EventFlow's input requirements. The goal of EventFlow's built-in transformation and simplification capabilities is to allow users to cast a wide net in the extraction process. This can help them to avoid the errors that are inherent in the specification of complex command-based queries. The focus of the initial extraction should revolve around the following three features:

1. Necessary event categories.
2. Relevant window of time.
3. Records of interest.

Users should try to capitalize on any extent to which the first two features dictate the third because it is comparatively easier to construct command-based queries against those first two features. For example, an initial extraction might be for all patients who took a blood thinning medication and went to the hospital in January 2006. EventFlow can then be used to narrow in further on the patients of interest. For example, maybe the only patients of interest are the ones who started taking the medication within a week of being discharged from the hospital, without taking any blood thinning medication while they were hospitalized. This type of more detailed criteria is better specified in EventFlow, where users get visible feedback about both the matching and non-matching records, as well as unexpected patterns in the overall data.

Step 2: Segment Records

The bulk of EventFlow's controls revolve around intra-record transformations and simplifications. While this includes the ability to shorten existing records, EventFlow does not allow users to segment an existing record into multiple records. This step must be accomplished before the data is loaded into EventFlow. The ideal segmentation is the one that produces the shortest records. For example, in the basketball dataset discussed in Section 6.7, the initial intuition was that a single game should correspond to a single record. However, this segmentation would produce extremely long records. By segmenting each game into two-possession increments, I was able to produce shorter records without losing critical information about event interactions. Similarly, medical datasets that track patients across years of time can sometimes be segmented into multiple month-long or week-long increments. Once a dataset can be analyzed and understood at the finest level of segmentation, users can more easily step up to coarser segmentations.

Step 3: Load Data into EventFlow

The initial display of a dataset will allow users to determine how their raw data corresponds to their research objectives. However, it may be the case that the initial dataset is too large to load into the application. The ultimate solution to this problem is to rebuild EventFlow with a database back-end, as described in Section 8.2. Until that change is made however, there are three sequential steps that users should consider taking (typically in a spreadsheet tool such as Microsoft Excel) to effectively load their dataset:

1. Remove record-level or event-level attributes.
2. Remove non-essential event categories.
3. Remove a random subset of records.

Once the data can be loaded into EventFlow and properly evaluated (see Step 6), additional transformations in EventFlow can frequently reduce the size of the dataset to the point at which initially omitted information can be added back in (see the study described in Section 6.8).

Step 4: Evaluate Segmentation and Extraction

While EventFlow's data transformation capabilities are designed to help reduce the complexity of, and thus the errors that occur during, the data extraction process, errors can still occur when users are preparing their initial dataset. Even something as simple as a keystroke error can produce massive inconsistencies in the resulting data. Fortunately, these errors are typically easy to spot when the data is loaded into EventFlow and a few simple operations have been performed. Large, obvious errors can typically be corrected quickly by making simple adjustments to the extraction script. When these errors occur, users should return to Step 1.

This is the step of the transformation process that motivated the bulk of this thesis. During both the LABA case study and the Opioid Misclassification case

study, my collaborators were continually identifying errors in the segmentation and extraction. Without any ability to clean the data within EventFlow, they had no way to push forward from this step, and had to repeatedly return to Step 1 to perform cleaning operations during their command-based extractions. Evaluating the data extraction should involve three steps:

Step 4a: Split Groups

Many research objectives revolve around comparing two or more groups of records (see the case study described in Section 7.5). If the distinction between these groups is not dependent upon events that are obscured, either by errors made during the extraction process or inherent complexities in the data, then users should begin the transformation process by grouping the display. This step may not always be possible this early in the process, but this step is still important to do as soon as possible so that the visualization reflects the user's research objectives.

Step 4b: Align Data

Many research questions over temporal event datasets revolve around a key event in each record. If this event is not obscured by errors or data complexity, then the next step that users should take is to align the dataset around this key event. Alignment, as well as the grouping performed in Step 4a, provides users with a view of their data that more closely matches their research objective. During my case study with the PVC, a single alignment of the data is what initially convinced the researchers of EventFlow's value. Again, this option is not always possible at this stage, but when it is, it can dramatically help the clarity with which users approach future transformations.

Step 4c: Filter

Once the records have been extracted, segmented, loaded, and (potentially) aligned, users should begin by engaging with EventFlow's filtering features. This can be done by asking the following questions about the dataset, and taking the corresponding actions if the answer is "yes":

- Are certain event categories irrelevant to the questions I'll be asking? → Use legend to filter out event categories (see Sections 6.6 and 7.7).
- Is there a visible group of records that are irrelevant to the questions I'll be asking? → Select and remove records (see Sections 6.5 and 6.6).
- Do I only care about the events that occur immediately before or after the alignment point? → Set window by event count or time (see Sections 6.2 and 6.4.4).

Step 5: Basic Transformations

Once filtering has been performed, users should move on to the transformation features that EventFlow supports directly from the control panel and file menu. Again, these transformations can be assessed by asking the following questions:

- Are small gaps and overlaps between intervals of the same category irrelevant to the questions I'll be asking? → Merge intervals (see Sections 6.4.4 and 7.6).
- Do multiple event categories convey essentially the same information? → Aggregate event categories (see Sections 6.7 and 6.4.4).
- Is important information hidden in the event attributes? → Switch event category for attribute value (see Sections 6.7 and 6.2).
- Do I only care about the n^{th} event of a given category? → Add marker event at alignment (see Sections 7.6 and 7.5).

Step 6: Align, Group, Filter

If alignment and grouping were not previously possible, they should be attempted again at this point in the transformation process. The basic transformations may also reveal additional event categories or records that are no longer relevant, which can be removed from the dataset. Users should revisit Steps 4, 5, and 7.

Step 7: Advanced Transformations

The final round of transformations and simplifications should be performed using the Find & Replace system. The operations performed in this step are typically very domain and objective dependent, however, there are some overarching strategies that users should consider:

- Replace a sequence of events, such as repeated events of the same category, with a single, continuous interval (see Sections 7.4 and 7.8).
- Mark interesting points with records, such as where multiple intervals overlap (see Sections 7.7 and 6.4.4).
- Reclassify certain events as a new category, such as changing a "Doctor Visit" event to a "Follow-up Visit" event (see Sections 6.5 and 6.4.4).
- Remove extraneous events that occur before, after, or during a key event (see Sections 7.7 and 7.8).

Step 8: Iterate

Return to Step 4 if necessary.

Step 9: Analysis

Prior to this work, this step was the intended successor of Step 6. The goal of this work was to more efficiently bridge this gap so that users could use EventFlow to

arrive at their intended insights.

6.10 Summary

Event sequence transformation and simplification is critical to obtaining population-level overviews and more accurate representations of real-world events. This chapter reports on an in-depth design study that resulted in targeted transformation techniques, allowing users to precisely and iteratively pare down complex temporal event datasets to the key visual elements that reveal meaningful patterns. This work draws on techniques from both temporal event query and data mining, as well as countless hours spent with domain experts to understand how temporal relationships can be accessed and transformed within complex datasets.

These innovative transformation techniques could benefit many developers of temporal analysis systems as well as the researchers who use them. Working with 5 real-world user teams, simplification-based transformation reduced the visual complexity of initially overwhelming datasets by over 80%. This reduction allowed researchers to quickly and successfully generate and test hypotheses, as well as produce comprehensible figures for communicating their results. Event sequence transformation, as demonstrated here, is a powerful and generalizable approach for solving problems with temporal datasets.

While EventFlow has been successful thus far in allowing users to reduce complexity, it is important to remember that these capabilities make it equally possible for users to remove or obscure important features of their datasets. This can occur either accidentally or as a deliberate attempt to mislead others. To prevent this from occurring, EventFlow is coupled with a logging system, so that new datasets are generated with a record of the modifications performed.

Chapter 7

EventFlow Case Studies

Multi-dimensional In-depth Long-term Case Studies (MILCS) have become standard in the evaluation of how visual analytics systems can impact real world research questions [98]. While short-duration, in-laboratory studies can be instrumental in evaluating individual components of such a system (see Appendix B), these studies are typically conducted with non-experts or over artificial datasets. This is because it is impossible to replicate the domain expertise of a real-world scientist across a large number of test users, and it is impossible to constrain a potentially open-ended exploratory analysis to only a few hours of time. MILCS allow real domain experts to conduct analyses in their own environment and on their own schedule [80, 81, 96, 119]. In many cases, these MILCS can operate in what is referred to as "chauffeur" mode, where the researcher works directly with the software expert - in this case myself - to provide guidance on how best to utilize the software to achieve their objectives. This collaboration can streamline the initial process of learning the software. Overall, MILCS offer a more complete view into the insights and roadblocks that these researchers encounter.

The bulk of this dissertation stems from the failings of my early MILCS. At the beginning of this research, multiple case studies were stalled or postponed due to errors in the data extraction process that were only discovered when the dataset was initially loaded into EventFlow. While this can be lauded as the perfect example of how visual analytics can reveal unknown features of a dataset, the ultimate result was that my case study partners were not reaching the desired stage at which EventFlow could be used to address their primary research objectives. Looking back at the design study framework described by Sedlmair et al. [95], these early

collaborations had all of the components necessary to arrive at the deployment stage, which is well within the core phase of the design study framework, but the deployments were continually revealing errors and complexities in the data that prevented the studies from moving into the analysis phase. Many collaborators would fall of touch or simply drop out during this lengthy process of iteratively cleaning the dataset through extraction.

My ultimate discovery was that these early setbacks were due to a lack of visual support in the data extraction and cleaning process. Researchers were extracting their data from large databases, using highly complex sets of criteria, but were receiving virtually no informative feedback on whether these operations were performing as expected. It was only when the resulting datasets could be seen in their entirety that researchers began to notice mistakes and oversights in their data extraction process. As a result of these initial observations, EventFlow was re-imagined as not only a tool for visual exploration, but also a tool for performing carefully designed transformations. My goal was to push the capabilities of the software as close as possible to the raw data, so that the visualizations could be leveraged to more accurately perform extraction and cleaning steps. This change eliminated the late-stage case study failures that I was initially observing. It also expanded the potential user base from "users with perfect data" to "users with data." This latter group, I found, comprises the vast majority of real-world researchers.

This chapter reports on eight case studies, summarized in Table 7.1, that illustrate the use of EventFlow in performing both data cleaning and data analysis in real world settings. In many cases, the outcomes and insights that were achieved in the data analysis stage would not have been possible without the preceding data cleaning stage. Being able to perform both of these tasks within a single application allowed EventFlow to play a much more prominent role in the overall research process. Where possible, I report the detailed metrics of how the data cleaning phase impacted the eventual dataset that was used for analysis. In cases where

I was not given direct access to the dataset, I provide screenshots that illustrate the transformations that were performed. The case studies not reported in this chapter are those that failed in the extremely early stages (typically due to logistical issues, or missing requirements on the users' end), those that only required minimal transformations to arrive at the intended insight, and those where my involvement was too minimal to fully relate the users' process.

7.1 Pediatric Trauma Procedures

The Pediatric Trauma Procedures case study is described in detail in Section 6.6. Provided here is summarizing information, as well as additional information not included in the previous section.

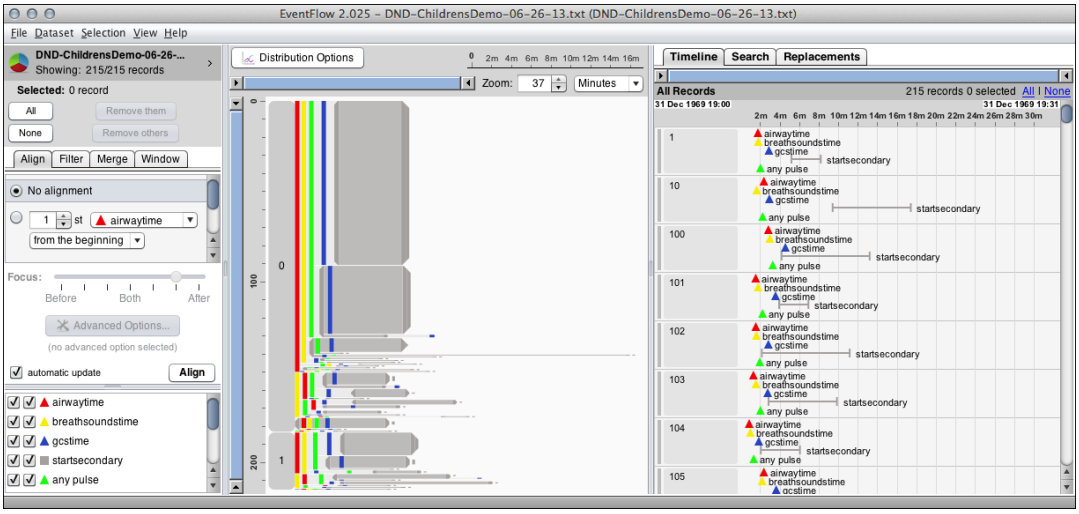


Figure 7.1: Pediatric trauma procedures case study.

Partner: Beth Carter, Research Scientist, Division of Trauma and Burns

Organization: Children's National Medical Center

MILCS Level: Mature, Chauffeur Mode

Duration: March 2012 - November 2013

	Title	Domain	Dataset	Duration	Level	Mode
7.1	Pediatric Trauma Procedures	Medical	111 patients	1.5 years	Mature	Chauffeur
	Outcome: Found that deviations from the ATLS protocol occur frequently and in many forms.					
7.2	LABA Guideline Adherence	Medical	100 patients	2 years	Mature	Chauffeur
	Outcome: Determined that doctors were not adhering to step-up/step-down prescription guidelines.					
7.3	Opioid Use Classification	Medical	1000 patients	2 years	Mature	Chauffeur
	Outcome: Confirmed accuracy of opioid use classification.					
7.4	Warfarin & Traumatic Brain Injuries	Medical	1634 patients	5 months	Mature	User-Driven
	Outcome: Characterized warfarin use patterns surrounding a traumatic brain injury.					
7.5	Radiation Classification	Medical	9826 patients	5 months	Mature	User-Driven
	Outcome: Developed an algorithm for discerning radiation treatment types in claims data.					
7.6	Total Parenteral Nutrition	Medical	1000 patients	8 months	Mature	User-Driven
	Outcome: Determined the frequency and development of liver disease in patients who received TPN.					
7.6	Trauma Role & Activity	Medical	55 patients	10 months	Early	Chauffeur
	Outcome: Identified and characterized patterns of behavior during trauma resuscitations.					
7.8	Debugging Distributed Storage Systems	Software	3792 blocks	4 months	Mature	User-Driven
	Outcome: Evaluated the performance of the system during normal operation and after crashes.					

Table 7.1: Summary of EventFlow case studies.

Data: The patient-centered tasks performed by the trauma team when a patient is brought to the trauma bay.

Goal: Understanding trauma team adherence to ATLS protocol.

Records at start: 215 patients

Average events per record at start: 19 events

Visual complexity at start: 1209 visual elements, .84% of display height per element

Records at end: 111 patients

Average events per record at end: 5 events

Visual complexity at end: 60 visual elements, 11.58% of display height per element

Data Cleaning:

1. Data reformatted from a horizontal event listing (event categories listed in a single, top row, and each patient listed as an additional row) to the EventFlow format.
2. Secondary survey interval derived from independent end points.
3. Extraneous event categories filtered out.
4. Category colors changed to make the correct ordering more visible.
5. Two pulse check categories merged into a single category.
6. Find and replace to include only one pulse check per record.
7. Removed records with correct sequence.

Study Procedures

The case study began at Children's National Medical Center, with a tour of the trauma bay, and a walkthrough of the procedures that take place, the personnel that are involved, and the roles that they play. The trauma bay is equipped with cameras that record each trauma that is brought in. The dataset was compiled by members of the research team, who would watch these videos and record the timestamps of the events that took place. It most cases, these timestamps were identified through audio by members of the trauma team calling out their actions as they performed them. I was shown an example video to demonstrate the event recording process.

Four subsequent meetings were conducted with Dr. Carter, each lasting between one and two hours. The first of these took place about two months after our initial meeting, and revolved around reformatting Dr. Carter's dataset to match the EventFlow input format. In the following two meetings, Dr. Carter and I explored and transformed the dataset. Dr. Carter dictated her evolving questions, and I performed the analogous operations in EventFlow. In our final meeting, Dr. Carter was given control of EventFlow, and was able to duplicate our previous work on her own as I watched on. The goal of this final meeting was for her to be able to demonstrate the entirety of our study to her colleagues.

Outcomes For User

This case study resulted in three unexpected findings. The first was the small percentage of records that adhered to the ATLS protocol. The second was in the variety of sequences that did not match the ATLS protocol. Dr. Carter had expected to see a small number of very common deviations from the protocol. In fact there were 29 unique deviations. EventFlow not only revealed this result, but made it easy for Dr. Carter to communicate about it with the rest of her team. Finally, Dr. Carter and the trauma team has long assumed that deviations from

the ATLS were primarily caused by insufficient prep time, a condition that was beyond their control. However, EventFlow clearly showed that this condition had no significant effect on the trauma team's adherence to the protocol. Dr. Carter presented this work at the 2013 American Medical Informatics Association conference [11], and secured funding to continue the collaboration.

Outcomes For EventFlow

This case study was unique in that it was completed without necessitating any new software features or extensions. Because of this, it highlighted the potential of EventFlow to serve just as effectively as a data cleaning tool as it could as a data exploration tool. This was the first data cleaning process that was conducted from start to finish with the fully-functioning Find & Replace feature. It offered the first look at what this process looked like without being tightly coupled with an interactive, feature design process.

This was also the first study to visibly benefit from EventFlow's flexible event coloring scheme. The software allows users to set the display of each event category to the color of their choosing. This feature was built into the original LifeFlow tool, but had primarily been used to highlight a particular category, or in cases where the number of event categories exceeded EventFlow's list of default colors. In this case, the coloring scheme was used to better highlight event ordering, which is one of the primary, overall goals of the software. Patients who were treated according to the ATLS protocol were extremely visible in the dataset simply because that event sequence produced a proper rainbow in the display. In subsequent studies, coloring schemes became a more tangible checkpoint in the data cleaning process, whereas before it was only considered in cases where there was an obvious coloring problem.

Limitations

This case study highlighted the tradeoffs that occur when the data collection process is closely tied to the analysis objectives. Dr. Carter's goal of evaluating adherence to the ATLS protocol was known prior to the data collection process and, as such, the data was explicitly collected in a manner that would support this analysis. It is critical to note that, even under these circumstances, EventFlow's filtering and data transformation tools were still necessary to address Dr. Carter's research objectives. These data transformations, however, were far less extensive and complex than those typically needed to support studies where the data collection is done without any knowledge of or dependence upon the analysis objectives.

There are, however, disadvantages of designing the data collection process around the overarching research objective because the dataset will not necessarily support new questions that evolve from unexpected results. For example, due to the unexpected variety of deviations from the ATLS protocol, Dr. Carter was interested in whether certain deviations were more likely to result in negative outcomes than others. However, information about patient outcome had not been incorporated into the original data collection. While EventFlow can perform a wide variety of data transformations and re-representations, it does not support analysis over missing data. This case study is a rather extreme example, in which the missing data represents a complete unknown, but overall, EventFlow's Find & Replace system could be augmented to allow new events to be inserted in places that do not exactly correspond to existing events. This functionality would allow researchers to better account for missing data, as well as misrepresented data.

Summary

The Pediatric Trauma Procedures case study was, and continues to be, the most straightforward and relatable of the studies I conducted. It served as the best example of how extraneous clutter and seemingly harmless nuances of the data collection

process could significantly obscure meaningful event patterns. In this study, the event pattern of interest was singularly defined, which allowed us to transform the dataset with extreme precision, ultimately resulting in a set of records and events that exactly mirrored the objective of the study. This case study thus served as an ideal way to demonstrate the use of EventFlow to new users.

7.2 LABA Guideline Adherence

The LABA Guideline Adherence case study is discussed extensively throughout Section 4.4 and Chapter 6. Provided here is summarizing information, as well as additional information not included in these previous chapters.

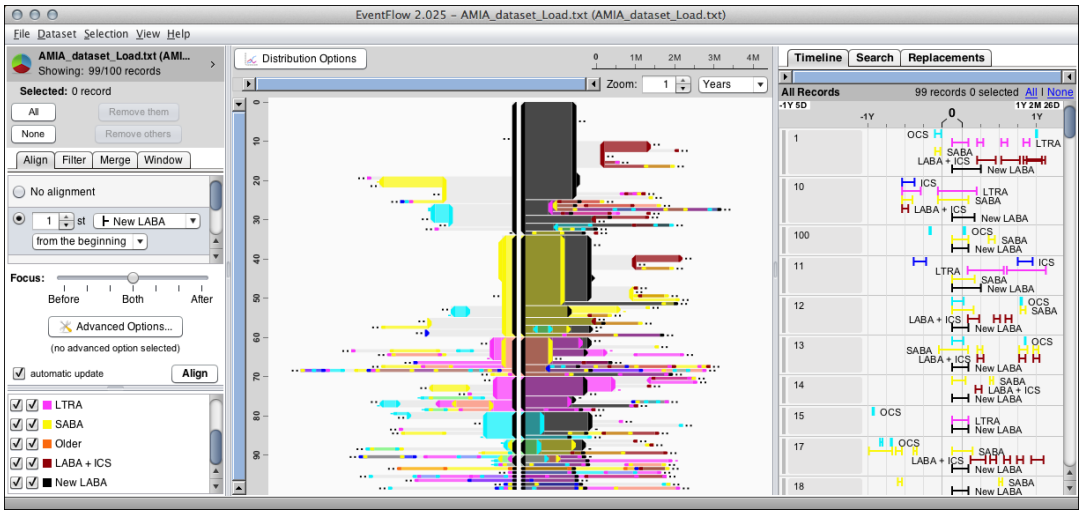


Figure 7.2: LABA guideline adherence case study.

Partner: Tamra Meyer

Organization: U.S. Army Pharmacovigilance Center, Department of Defense

MILCS Level: Mature, Chauffeur Mode and User Driven

Duration: September 2011 - November 2013

Data: The asthma-related prescriptions of patients in the military health system

who received a new LABA prescription between January 2006 and January 2011.

Goal: Determine whether physicians were prescribing LABAs according to FDA guidelines.

Records at start: 100 patients

Average Events per Record at Start: 16 events

Visual Complexity at Start: 2700 visual elements, 1.14% of display height per element

Records at end: 100 patients

Average Events per Record at End: 5 events

Visual Complexity at End: 491 visual elements, 2.08% of display height per element

Data Cleaning:

1. Add marker event for "new LABA prescriptions.
2. Filter out extraneous categories.
3. Replace concurrent LABA and ICS prescription with single LABA + ICS interval.
4. Replace LABA prescription marked as "new with a single New LABA interval.
5. Merge intervals to reflect exposure.
6. Aggregate similar medications into single category.
7. Align by point of interest.
8. Filter display to the six months on either side of the alignment.

Study Procedures

This study began in September 2011 with a meeting to introduce members of the Pharmacovigilance Center to the EventFlow software. The EventFlow team met with a group of epidemiologists (including Dr. Meyer), statisticians, and data scientists, as well as the PVC's leader, Colonel Trinka Coster. They were shown a series of demos to highlight EventFlow's then newly implemented interval support. They were also given a brief tutorial on the EventFlow data format. Dr. Meyer was introduced as one of the primary case study candidates within the group. She was conducting a project for the U.S. Food and Drug Administration (FDA) to better understand adherence to LABA (long-acting beta-agonist) prescription guidelines within the Military Health System. She was looking for two primary patterns that would be indicative of doctors adhering to these guidelines. First, a LABA should always be prescribed in tandem with an inhaled corticosteroids (ICS). Second, patients who were prescribed a LABA should have recently been prescribed another, less-potent asthma medication. That is, a LABA should only be prescribed when other, less-potent treatments have not worked. Finally, once a LABA prescription is finished, patients should return to one of these less-potent asthma treatments. This prescription pattern is referred to as "step up, step down."

As alluded to previously, the data for this study came from the Military Health System, which serves 9.5 million active duty and retired military personnel as well as their families. This study focused on the 182,498 patients who were prescribed a new LABA prescription between January 2006 and January 2011. This period of time directly followed the dissemination of the FDA's LABA prescribing guidelines. Thus the goal was to not only understand the overall adherence to the LABA guideline, but to also see if this adherence was improving over time. Prior to using EventFlow, Dr. Meyer had been conducting this study using the command-based SAS software, which produced virtually no visual output.

Over two years, I met with Dr. Meyer on over a dozen occasions. Our meetings lasted approximately two hours and were typically held once every two months. She routinely worked on sample datasets of 100 patients. Since a single patient record could contain years worth of medication history, this sample size made it easier to spot both bizarre and meaningful outliers. Dr. Meyer's datasets also evolved dramatically over this two-year period. In the majority of my initial meetings with Dr. Meyer, EventFlow revealed obvious errors in the dataset, either due to a SAS scripting error or a condition that simply hadn't yet been considered. As a result, Dr. Meyer was able to continually refine her data extraction scripts until it was producing the correct results. This phase of the case study was critical, as these errors and oversights had previously gone unnoticed. Dr. Meyer was also able to narrow in on more specific questions, using both SAS and EventFlow. For example, two sessions were spent using EventFlow to determine whether she had chosen the best criteria for what constituted a "new" LABA prescription for a given patient. Since a patient's "new" LABA prescription was the anchor point around which the entire study revolved, it was critical to be sure about this prescription.

In addition to our individual sessions, Dr. Meyer presented her evolving working at both an internal EventFlow user group meeting and at the Human-Computer Interaction Lab's annual symposium. These presentations allowed her to get feedback on her use of EventFlow from other experienced users, as well as serve as an example to new users who were just getting started with the software. Dr. Meyer presented her final work at the AMIA 2013 workshop on visual analytics in healthcare [69]. Her presentation was described by other attendees as one of the best practical demonstration of how visual analytics can be used in a real research setting.

Outcomes For User

Dr. Meyer was able to conclude that doctors were consistently adhering to the guideline of always prescribing a LABA in tandem with an ICS prescription. How-

ever, there was far less consistency in the adherence to the step up/step down protocol. These results were confirmed against the entire dataset using SAS queries. Dr. Meyer also gained a better understanding of how doctors were deviating from the step up/step down protocol, and how frequently it was happening. For example, about a quarter of the sample dataset had been prescribed a new LABA without the appropriate step up prescriptions.

EventFlow was also used extensively as a communication tool between members of the PVC, members of the FDA, and the doctors they work with. The software shut down the office during one of my initial sessions with Dr. Meyer as the entire PVC staff clustered around the computer to "see" the data for the first time. This LABA study had been going on long before EventFlow was introduced, and the researchers found it hugely rewarding to finally be able to see the trends that they had been discussing and speculating over for months. One doctor, with whom Dr. Meyer was collaborating, immediately noticed an alarming prescription pattern that appeared as an outlier in EventFlow. He concluded that patients with this pattern should be flagged for immediate intervention.

Dr. Meyer ultimately gained a better understanding of the event patterns that existed in the dataset. She had been using SAS to isolate patients with what she considered to be the ideal step up/step down pattern. However, using EventFlow, she was able to identify additional patients who did not have this ideal event pattern, but who had still been prescribed asthma medications according to the FDA guidelines (see Section 4.3). These additional patients almost doubled the set of patients who received the correct prescriptions, and changed the initial perception of the overall dataset.

Outcomes For EventFlow

Nearly every feature implementation, as well as my broadening understanding of visual analytics over temporal event data, was rooted in this case study. The collaboration with the PVC is what initially motivated the support of interval events

in EventFlow, which is ultimately the root of this dissertation. The researchers at the PVC, along with Sigfried Gold (an outside collaborator), made the compelling argument that their questions could not be answered using only point-based event representations. Their questions and urgings are what spurred the transformation of the LifeFlow software into EventFlow. From there, this case study was the inspiration for multiple software features including:

- Filter by Time Range
- The Overlap Filter
- Marker Event Insertion
- Interval Merging

The bulk of these features went on to motivate EventFlow's Find & Replace system, which Dr. Meyer was the first to test. This case study also helped to codify my own understanding of the process that takes place between data collection and data analysis. Even using small, 100-patient sample sets, the amount of unnecessary clutter in the LABA datasets, and the degree to which it hindered Dr. Meyer's ability to answer seemingly simple questions, are what prompted a shift in focus to allow users to transform the underlying data rather than simply explore it.

Limitations

The most tangible limitation of the LABA case study was that it informed more features than it actually benefitted from. That is to say, EventFlow perhaps benefitted more from this case study than Dr. Meyer did. While command-based queries and data transformations are time and labor intensive, they can still be performed faster (particularly by an expert user) than building these features from scratch. Even accounting for the time it took to identify and correct for errors in these

command-based transformations, Dr. Meyer was typically able to execute these transformations faster than I was able to prototype the graphic-based equivalent. As a result, there were many findings that could be reproduced in EventFlow, typically within a few short hours, but could not be directly attributed to EventFlow and are thus not reported in this thesis.

Summary

The LABA Guideline Adherence was the most long-term and in-depth case study I conducted. It served as the best example of how visual analytics can be used not just to answer existing questions, but to generate new ones. Many of my sessions with Dr. Meyer were spent exploring a single quirk in the dataset, or reevaluating the initial assumptions and specifications of the study as a whole. This study was the first to demonstrate that some datasets are much better suited to the EventFlow visualization than others. However, the ultimate finding was that these latter datasets - the less ideal ones - could be transformed into highly usable datasets without losing or obscuring critical information.

7.3 Opioid Use Classification

The Opioid Use Classification case study is discussed extensively throughout Sections 4.4 and 6.5. Provided here is summarizing information, as well as additional information not included in these previous sections.

Partner: Rose Thelus

Organization: U.S. Army Pharmacovigilance Center, Department of Defense

MILCS Level: Mature, Chauffeur Mode and User Driven

Duration: September 2011 - November 2013

Data: Patients with opioid prescriptions in the Military Health System

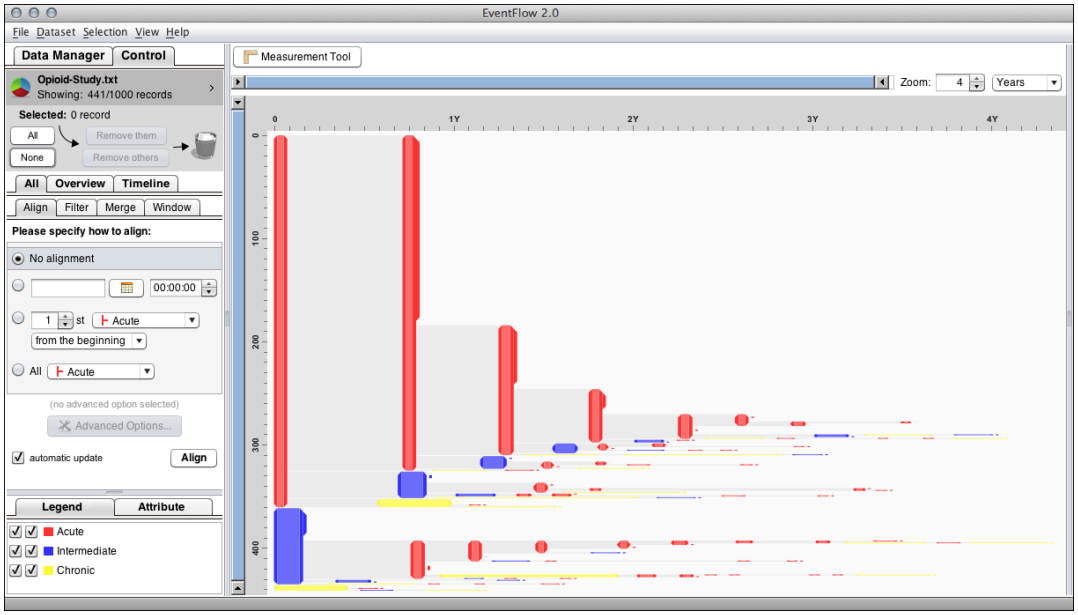


Figure 7.3: Opioid use classification case study.

Goal: To classify different types of opioid use as a means of identifying patients with habit-forming use patterns.

Records at start: 1000 patients

Average Events per Record at Start: 2.6 events

Visual Complexity at Start: 1064 visual elements, .49% of display height per element

Records at end: 441 patients

Average Events per Record at End: 1.7 events

Visual Complexity at End: 156 visual elements, 3.79% of display height per element

Data Cleaning:

1. Mark first prescription of each episode.

2. Replace $<$ 60 day episodes consisting of two or fewer prescriptions as an

acute episode.

3. Replace $>$ 90 day episodes as a chronic episode.

4. Replace 60-90 day episodes consisting of at least 10 prescriptions as a chronic episode.

5. Replace all remaining episodes as intermediate episodes.

6. Remove correctly classified acute patients.

7. Filter to only patients with consecutive acute episode.

Study Procedures

The opioid use classification study began at the same meeting as the LABA guideline adherence study, where EventFlow was introduced to the entire PVC team. Again, Dr. Thelus was working on data from the Military Health System. She began with a sample dataset of 500 patients, and gradually moved up to a final sample size of 1000 patients. I met with Dr. Thelus on approximately fifteen different occasions, during which Dr. Thelus and I took turns driving the software depending on the complexity of the task being performed. The bulk of these meetings took place in bi-weekly increments between September 2012 and December 2013, when Dr. Thelus was conducting the bulk of her study. The remainder of our meetings were held when Dr. Thelus had a specific questions about her data or the software. Our meetings typically lasted one hour.

The study took place in two phases. In the first phase, Dr. Thelus was interested in the dosage strength of prescriptions that were administered during both acute and chronic episodes. This exploration revealed an unexpected number of patients who had been prescribed a high-dosage opioid as part of an acute episode. These patients had no preceding hospital visits, which might have explained this prescription strategy. This finding led to the second phase of the study, where Dr.

Thelus reevaluated her criteria for classifying acute and chronic opioid episodes. Her hypothesis was that these high-dosage prescriptions might be part of misclassified chronic episodes. Ultimately however, she determined that these episodes had indeed been classified correctly, and that the high-dosage prescriptions during acute episodes were a potentially habit-forming trend.

Outcomes For User

Dr. Thelus had previously been using SAS, a command-based language, to classify patients as either acute, intermediate, or chronic users. She was doing this classification based on the standard criteria for making this determination, but otherwise had no further knowledge of the event patterns that existed in these records. Doing this classification in EventFlow though, allowed her to see every prescription pattern that was both included and excluded as she stepped through each condition. She reported that this process gave her much more confidence in the accuracy of the classification, and allowed her more freely explore the different patient groups.

Outcomes For EventFlow

The opioid use classification study was the primary motivator for two EventFlow features: category aggregation, and category-attribute switching. Dr. Thelus was constantly shifting her perspective between high vs. low opioid doses, acute vs. chronic opioid episodes, and acute vs. chronic patients. She had created three different datasets to represent these perspectives, as well as an aggregated dataset that contained event categories for all three perspectives. Overall, these strategies were cluttered and time-consuming, and it became clear that being able to perform these shift more gracefully would offer a substantial advantage. The category-attribute switching allowed information from different perspectives on the data to remain hidden until it was needed.

The category aggregation feature, which was also motivated by the LABA study, was critical in allowing similar prescriptions to be quickly grouped into a single medication class. This operation is easy to perform in SAS, however it produces an event category that cannot be ungrouped. Category aggregation was initially conceived of as the counterpart to category-attribute switching. However, as both of these case studies progressed, it became clear that both of the features were needed as stand-alone transformations that could be performed and undone independently.

Limitations

Much like the LABA case study, the Opioid case study demonstrated that while graphic-based approaches to query and transformation can be faster and more accurate than the command-based equivalents, an expert user can still typically execute these command-based queries faster than the same functionality can be prototyped in any alternate form. Because of this, these early stage case study partners did not benefit quite as much from the case study process as my later-stage case study partners did.

Summary

This study was the first to demonstrate the importance of visualization in classification. Specifically, the queries performed throughout this study highlighted the importance of users being able to see and understand both matches and non-matches, and to easily toggle their focus between these two groups. I was also able to see firsthand how the lack of confidence in command-based query results could have a tangible impact on a study. Even in this case, where the command-based queries had indeed produced the intended results, it was difficult for Dr. Thelus to move on to subsequent questions without being sure of these initial results. The confidence gained from visualizing the patient classifications was a critical step in

moving on to more complex queries, and continued to be a prominent theme in my subsequent case studies.

7.4 Warfarin and Traumatic Brain Injuries

Warfarin is a blood thinning medication that is frequently prescribed to elderly patients with atrial fibrillation (AF). The goal of the medication is to both treat and prevent blood clots that can lead to strokes. However, elderly patients are also increasingly prone to traumatic brain injuries, which can cause excess bleeding in the head. This excess bleeding, combined with a blood thinning medication like warfarin, can be extremely dangerous. It is typical for AF patients to stop warfarin use for a brief period of time after suffering a traumatic brain injury, however there is no consensus on the management of warfarin use beyond that. It is important for patients to resume warfarin use as soon as possible to manage the risk of a stroke, but this objective must be balanced against the risk of bleeding when warfarin use is resumed too quickly. Doctors are tasked with making this decision on a case by case basis.

My case study partner, Xinggang Liu, was interested in the decisions that doctors were making about resuming warfarin use after a traumatic brain injury. To do this, he compiled a dataset of 1634 patients who:

- Had suffered a traumatic brain injury between 1/1/2006 and 12/31/2009
- Were 66 years old or older
- Had been diagnosed with AF at least one year prior to the traumatic brain injury.

Based on the date of the traumatic brain injury, Dr. Liu looked at the period of time before the injury and the period of time after the injury in 30-day increments. The number of days with warfarin coverage within each of these increments (based

on prescription refills) was then used to classify each increment as either "With Warfarin," "Without Warfarin," or "No Warfarin Information." EventFlow was used to visualize patterns of these three event types (see Figure 7.4).

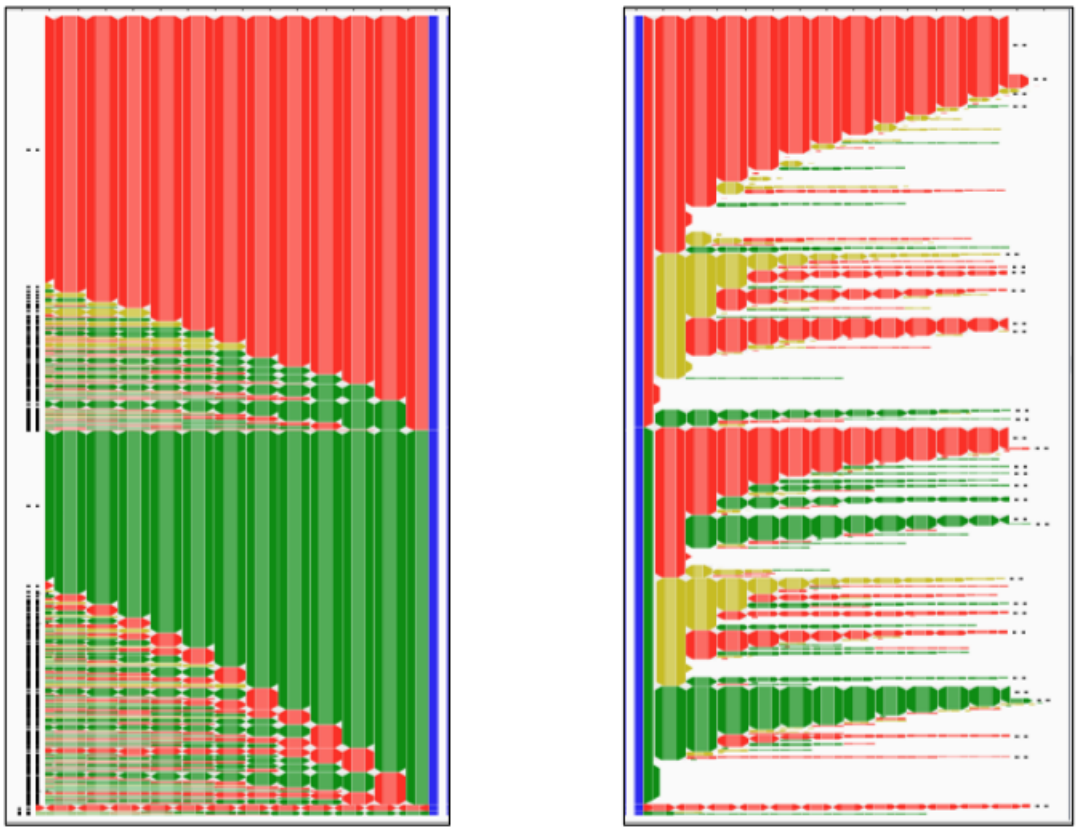


Figure 7.4: Left - Warfarin use patterns in the 12 months leading to the patients' traumatic brain injury. "With Warfarin" months are in green, "Without Warfarin" months are in red, "No Warfarin Information" months are in yellow, and the traumatic brain injury is shown in blue. Right - Warfarin use patterns in the 12 months following the patients' traumatic brain injury.

Partner: Xinggang Liu

Organization: University of Maryland School of Pharmacy

MILCS Level: Mature, User-Driven

Duration: June 2013 - October 2013

Data: Warfarin usage surrounding a traumatic brain injury.

Goal: To understand the patterns of Warfarin use for patients who sustain a traumatic brain injury.

Records: 1634 patients

Study Procedures

Dr. Liu was an extremely independent and self-motivated user. He was able to get his data formatted and up and running in EventFlow using primarily the EventFlow website and manual. From there, he and I conducted two meetings via screen sharing. He would describe either a question he had about the data or a difficulty he was having with the software, and I would walk him through one or more potential solutions. Of particular interest was using the advanced search to combine consecutive intervals of the same category. This transformation provided a different view of the data that focused more on the event patterns, rather than their occurrence in time (see Figure 7.5).

Based on our screen sharing meetings, Dr. Liu was able to conduct the bulk of his study on his own. However, he and I also met in person on three separate occasions to discuss the software and the various features and modification that would help Dr. Liu put the finishing touches on his study. These features (discussed below) were implemented prior to Dr. Liu's final presentation on October 31st, 2013, allowing him to show screenshots that better suited his purposes.

Outcomes For User

Because Dr. Liu was extremely motivated, I was able to see how much effort a user would have to expend in order to arrive at the types of insights that are easily gleaned from the EventFlow display. Dr. Liu, prior to seeing EventFlow, had created an extremely intricate Excel spreadsheet that showed each month of warfarin use as a colored cell. He had manually aligned these cells around the time

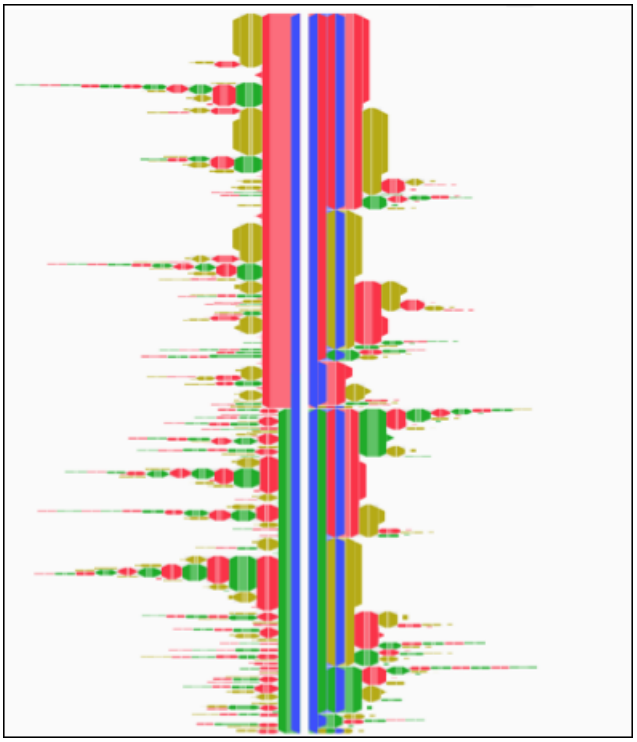


Figure 7.5: Warfarin use event patterns are better revealed when consecutive months of the same use are merged into a single interval.

of each patient's traumatic brain injury. Suffice to say then, he was extremely excited that EventFlow could produce a similar view in only a single alignment click. Much like my other case study users, Dr. Liu reported that the ability to see his data comprehensively dramatically increased his confidence in the results he uncovered.

Ultimately, Dr. Liu found that the prevalence of warfarin usage significantly decreased after a patient suffered a traumatic brain injury. While approximately 45% of patients had been using warfarin prior to their injury, only 17% of the patients were prescribed warfarin in the first month following their injury. This percentage increased in the subsequent months, finally leveling out around 30% after 12 months. Still though, this suggests that doctors may be acting in an overly cautious manner when starting or resuming a warfarin prescription after a patient suffers a traumatic brain injury. He concluded that policy makers and

practitioners should be aware and alerted to the increased risk of blood clotting in these patients. Dr. Liu presented these finding at the 66th Annual Scientific Meeting of the Gerontological Society of America [63].

Outcomes For EventFlow

The most significant outcome from Dr. Liu's work was a complete reworking of the way that EventFlow performs alignments. Previously the software would find the designated alignment point within each record, and essentially split the record at that point. Events preceding the alignment point would be processed into one aggregation, and events following the alignment point would be processed into another. These two aggregations were not linked in any way and, because of this, the horizontal continuity of a given record was not guaranteed to be preserved across the alignment point. Indeed, this horizontal continuity was very often not preserved. The alignment was being done this way because it allowed for the maximum amount of aggregation, and thus the most compact, readable, and memory-saving display. The EventFlow documentation clearly described this process and the effect it had on the resulting display.

In practice however, it was extremely difficult for users to read the aligned display without assuming that the horizontal continuity was being preserved. Though this effect had been observed in other case studies, this was the first time that it produced errors in the initial findings. This was due to the fact that Dr. Liu conducted most of his work in EventFlow on his own, and was not subjected to my constant reminders about this display quirk. It became clear though, that for EventFlow to be more widely distributed, users would have to be relied upon to conduct analyses without my help. The cleaner display was not worth the potential for erroneous observations that seemed inevitable given the current alignment strategy.

Because of this, the alignment was redesigned to include three additional options:

1. Only one side of the alignment is displayed.

2. Both sides of the alignments are displayed. One side of the alignment is constructed normally, and the other side is constructed to exactly preserve the horizontal continuity of each record from the focal side of the alignment.

3. Both sides of the alignment are displayed. One side of the alignment is constructed normally, and the other side is constructed such that every major group of records from the focal side of the alignment remains vertically aligned with the same group of records on the non-focal side.

4. The original alignment strategy, where each side of the alignment in constructed independently, remains as an available option for expert users.

Option 2 now functions as the default alignment and, when option 3 or 4 is selected, the user receives a pop-up message that explicitly states that the horizontal continuity of each record is not guaranteed to be preserved. While options 1-3 make one side of the alignment more difficult to read (see Figure 7.6), the user can always switch the alignment focus to that side in order to get a better look. Most importantly, the default alignment makes it impossible for users to make an erroneous observations about their data due to a lack of record continuity.

Limitations

While Dr. Liu performed the requisite data manipulations and transformations to satisfy his own interests as a domain expert, there were additional simplification and data cleaning steps that could have been performed in order to more clearly illustrate his conclusions to other users and non-experts. Part of this was likely due to Dr. Liu's hesitation to use some of EventFlow's more advanced data transformation controls without assistance. Also, when I am more closely involved in a case study, the perspective of a non-expert (in terms of the data domain) is naturally incorporated. This pushes users to make their findings more relatable.

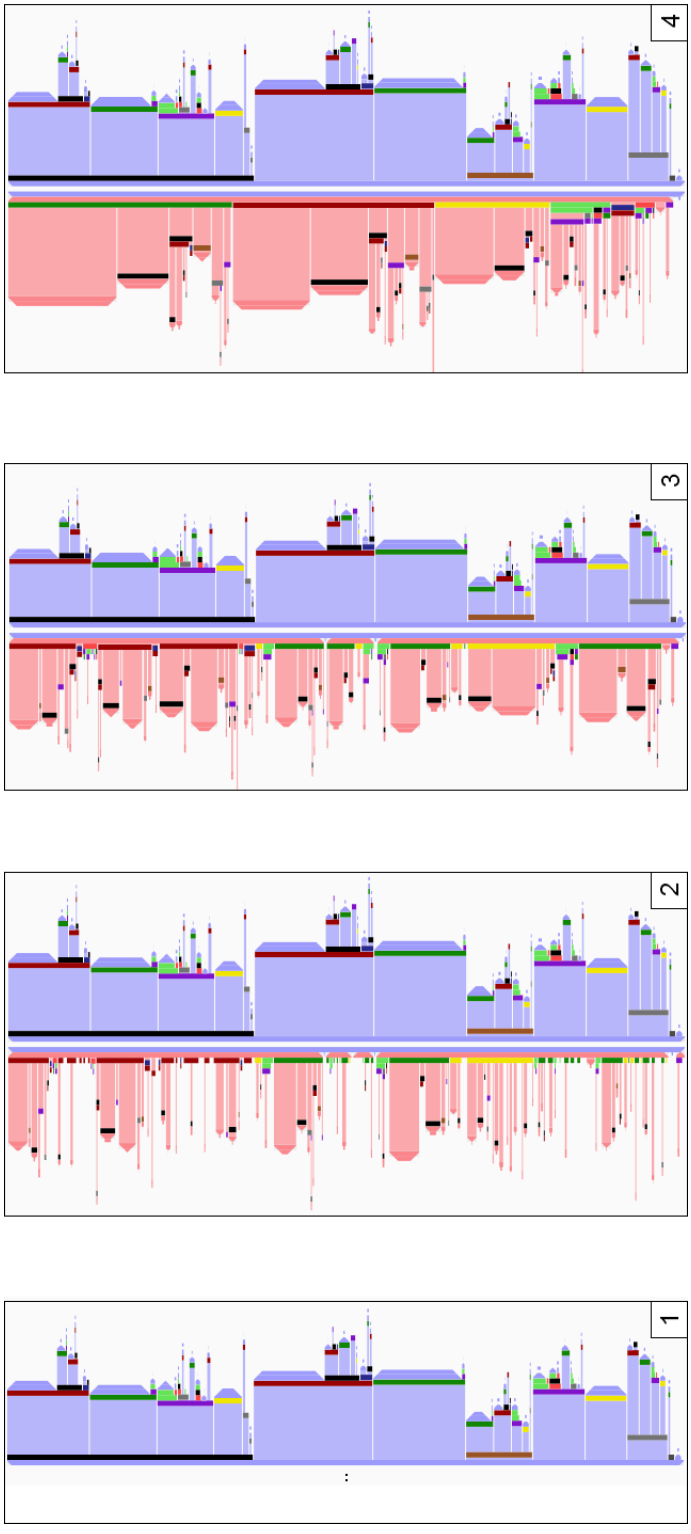


Figure 7.6: EventFlow's four alignment options, demonstrated on a basketball dataset. With the focus on the right side of the alignment (in blue), the left side of the alignment becomes more readable, moving from Option 1 (where the left side is not even displayed), to Option 4 (where it is displayed and fully aggregated).

In this case, the images that Dr. Liu used to convey his findings still required additional explanation so that non-experts could understand the trends that he was able to quickly spot (see Figure 7.7). While this is an unavoidable pitfall in terms of general use of the software, the lesson was that, when conducting case studies, my involvement can be valuable even if the users have already arrived at their conclusions.

Summary

The warfarin and traumatic brain injuries case study was the first case study that was conducted with minimal upfront involvement from myself, or other members of the EventFlow team. This had both positive effects as well as negative ones. On the positive side, this case study made it clear that a motivated user could learn the basics of the software without significant help, as well as arrive at valuable conclusions. This is a promising outcome as EventFlow is distributed to a wider base of users. However, one of the goals of EventFlow, and visual analytics in general, is to produce images and interactions that help to communicate findings to a broader audience. This perspective can be difficult to fully capture when the perspective of a non-expert is not incorporated into the analysis process.

7.5 Radiation Classification

The electronic health record (EHR) is, to a large extent, still coming into its own. As doctors and policy makers continue to work towards making the EHR the centerpiece of medical informatics, researchers frequently turn to insurance claims data to provide the most complete picture of a patient's treatment. The drawback of claims data, however, is that it focuses on billable events rather than the treatment details that might be clinically relevant. For example, almost 70% of patients with prostate cancer have bone metastasis. Very frequently, these patients are not only receiving radiation treatment to the prostate gland, but are also receiving

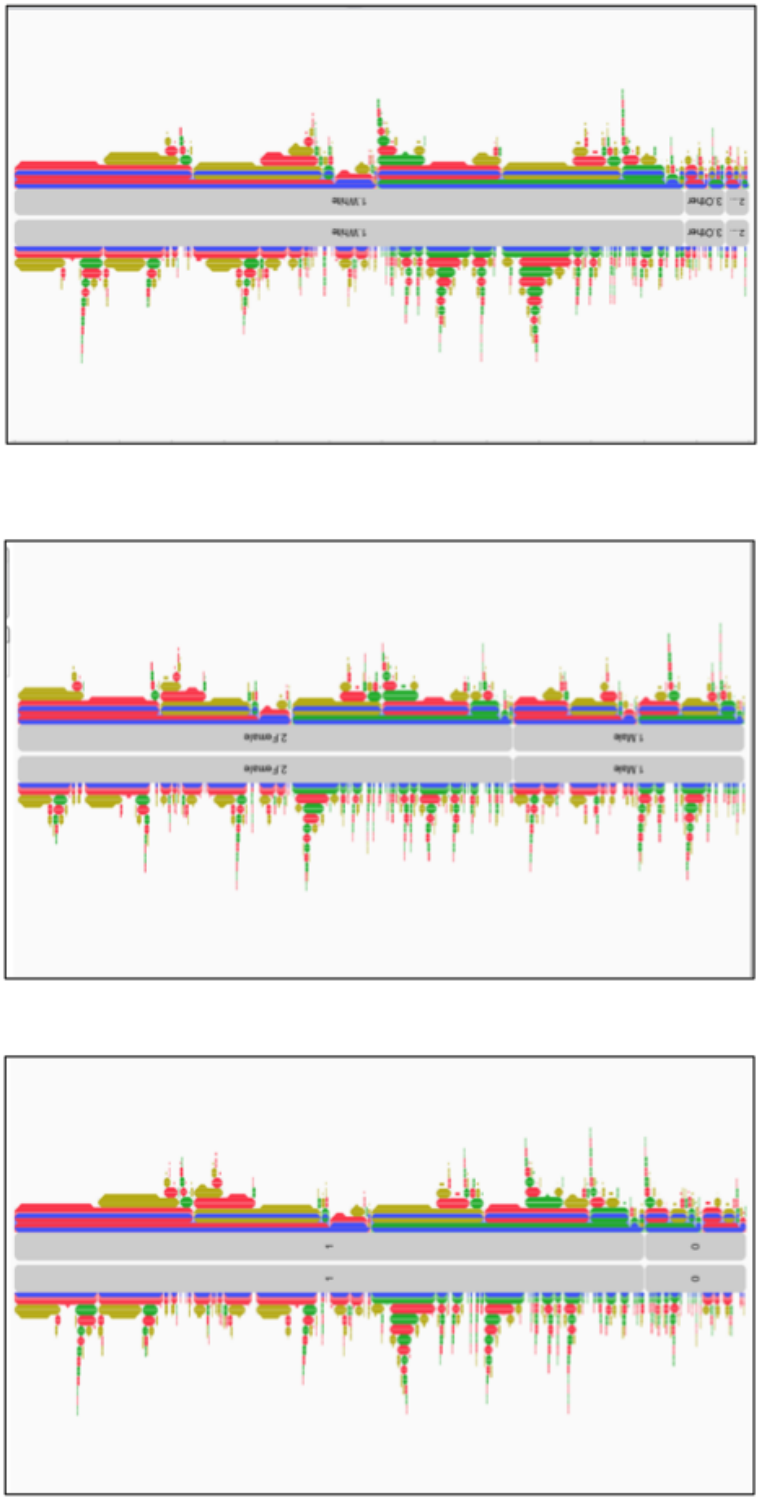


Figure 7.7: Warfarin usage broken down by age (left), sex (middle), and race (right). While the trends present in these figures are immediately apparent to a domain expert, they require additional explanation for non-experts. The trends could have been more clearly depicted by more extensively employing EventFlow's Find & Replace system.

radiation treatment to the bone. Claims data, however, uses the same ICD9 code for both of these radiation treatments (as they cost insurance companies the same amount). Because of this, researchers cannot distinguish between treatments to the prostate gland, and treatments to the bone.

Candice Young, a Ph.D. student at the University of Maryland School of Pharmacy, hypothesized that because of this inability to distinguish between prostate and bone radiation, that researchers may be overestimating the prevalence of radiation to the bone. Her goal was to develop an algorithm that could better classify radiation treatments based on the surrounding events in the claims data. Specifically, the course of radiation therapy to the bone is typically shorter than the course of radiation therapy to the prostate gland. Patients with bone metastasis are also prone to skeletal-related events such as pathological fractures and bone surgeries, which will appear in the claims data. She was interested in whether this variation in treatment duration and the proximity of skeletal events were enough to distinguish between radiation treatment types.

Partner: Candice Young

Organization: University of Maryland School of Pharmacy

MILCS Level: Mature, User-Driven

Duration: June 2013 - October 2013

Data: Treatments received by prostate cancer patients.

Goal: To develop an algorithm capable of classifying radiation targets based on event patterns in claims data.

Records: 9826 patients

Study Procedures

Much like my other case study with the University of Maryland School of Pharmacy (Section 7.4), this study was primarily user-driven. Ms. Young received only an initial introduction to EventFlow, and from there she was able to conduct the bulk of her analysis with minimal assistance. I met with Ms. Young on four occasions towards the end of her analysis, which allowed her to provide feedback about the software, and allowed me to answer any remaining questions that she had.

Ms. Young's data preprocessing incorporated information about radiation therapy duration, which allowed her to immediately identify these events without necessitating any data transformations in EventFlow. She also had a controlled dataset that was split into two groups: prostate cancer patients with bone metastasis, and prostate cancer patients without bone metastasis. This latter group of patients would not be receiving any radiation treatment to the bone, allowing Ms. Young to compare the event patterns in this group to the bone metastasis patients who would be receiving radiation to the bone. Ms. Young focused on the occurrence of short-duration radiation therapy in relation to two other event types: prostate cancer diagnosis and skeletal-related events (see Figure 7.8).

Outcomes For User

Ms. Young found that patients with bone metastasis had a higher prevalence of short-duration radiation therapy than patients without bone metastasis. Conversely, patients with bone metastasis had a lower prevalence of long-duration radiation therapy than patients without bone metastasis. She also saw that skeletal-related events tend to occur more often and sooner after the initial diagnosis in patients with bone metastasis. Finally, she found that patients with bone metastasis are frequently treated with short-duration radiation therapy within the 2 weeks after a skeletal-related event. Patients without bone metastasis have fewer short-duration radiation treatments following a skeletal-related event, and when they do

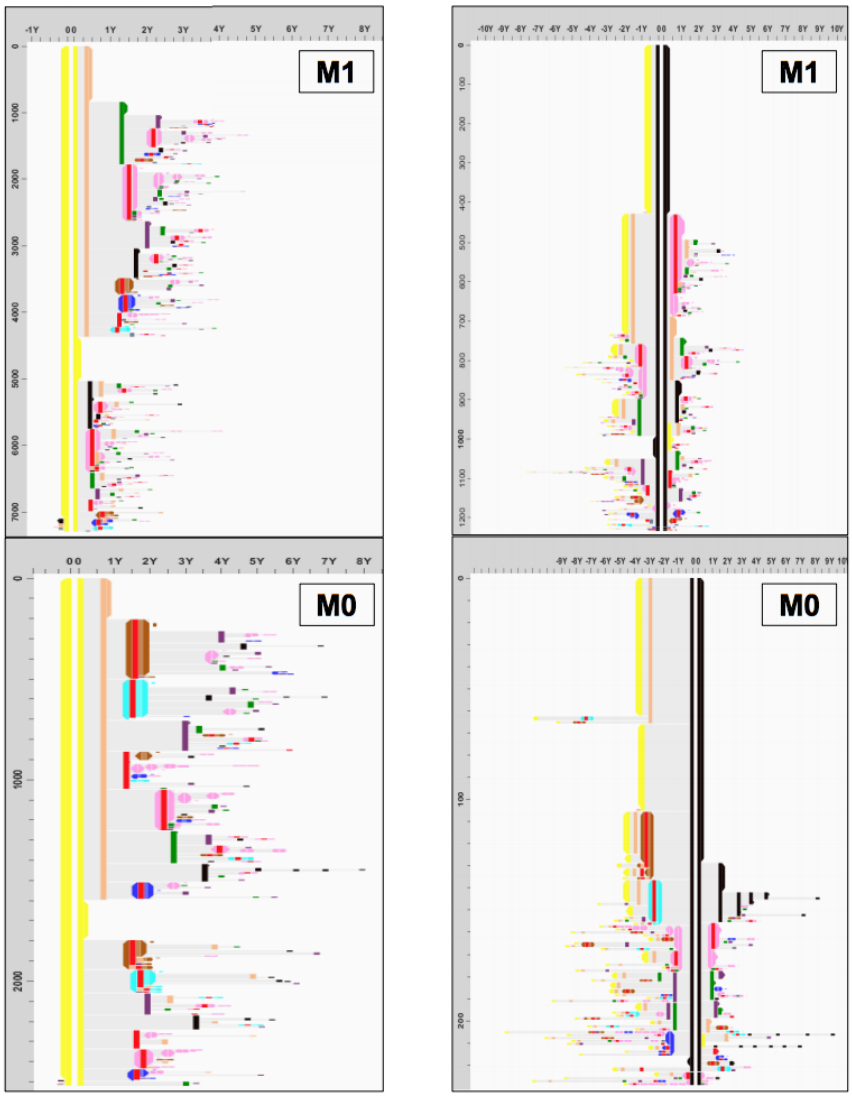


Figure 7.8: Patients were classified as having bone metastasis (M1) or no bone metastasis (M0). On the left, both patient groups are aligned by their prostate cancer diagnosis date (in yellow). On the right, both groups are aligned by their first skeletal-related event (in black). Short-duration radiation treatments are shown as a red point within a pink interval. Longer-duration radiation treatments are shown as a red point within a blue or brown interval. In the rightmost figures, Ms. Young noticed both the prevalence of the short-duration radiation treatment in M1 patients, as well as the proximity of this treatment to the initial diagnosis. In the leftmost figures, she saw that the short-duration radiation treatment closely followed a skeletal-related event in M1 patients.

occur, they typically occur more than 2 weeks after the skeletal-related event.

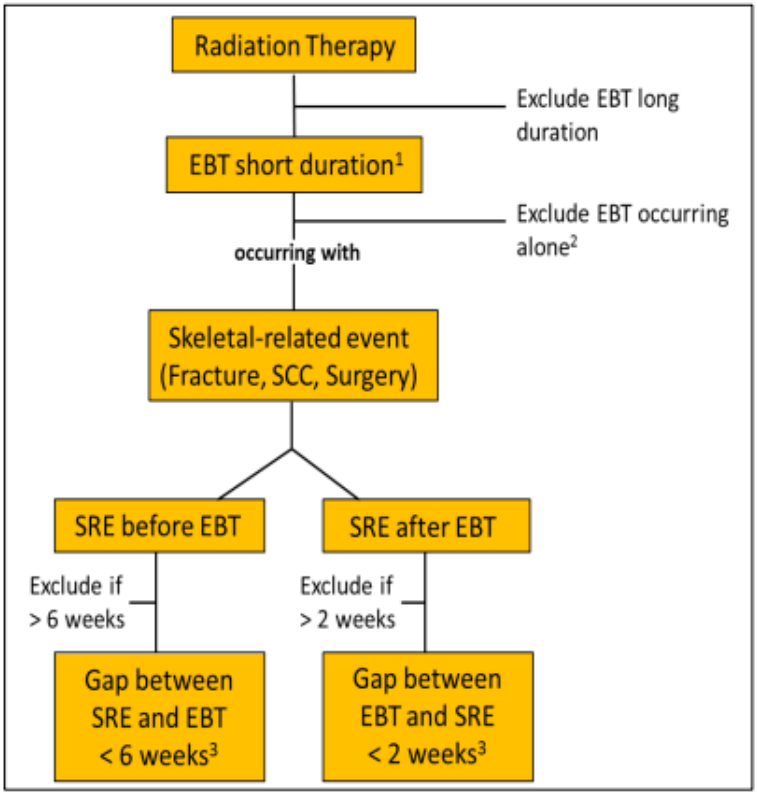


Figure 7.9: The algorithm that Ms. Young developed for identifying radiation to the bone in claims data.

Based on these findings, Ms. Young was able to develop the algorithm depicted in Figure 7.9. The algorithm incorporates information about event pattern frequency as well as timing. She hypothesized that the algorithm could be further enhance by including a bone metastasis diagnosis event in her dataset, and exploring the relationship of short-duration radiation therapy to this event. She focused her study on radiation treatments that are intended to cure bone cancer. However, she believed that the algorithm could be enhanced to identify radiation to the bone that is intended only to relieve symptoms. Ms. Young presented her findings at the 55th Annual Meeting of the American Society of Radiation Oncology [77].

Outcomes For EventFlow

EventFlow's aggregated view allows users to see every unique event sequence in a dataset. To do this, information about the time lapse between adjacent events is abstracted down to a single, averaged value. Users can then select a specific time lapse to see the abstracted details of each individual record. The problem is that when event sequences extend further and further away from the alignment point (which by default is the beginning of each record), the horizontal placement of event bars becomes the sum of these averages. Information about when an event occurs relative to the alignment point becomes more and more obscured as more and more events occur. Figure 7.10 illustrates this problem.

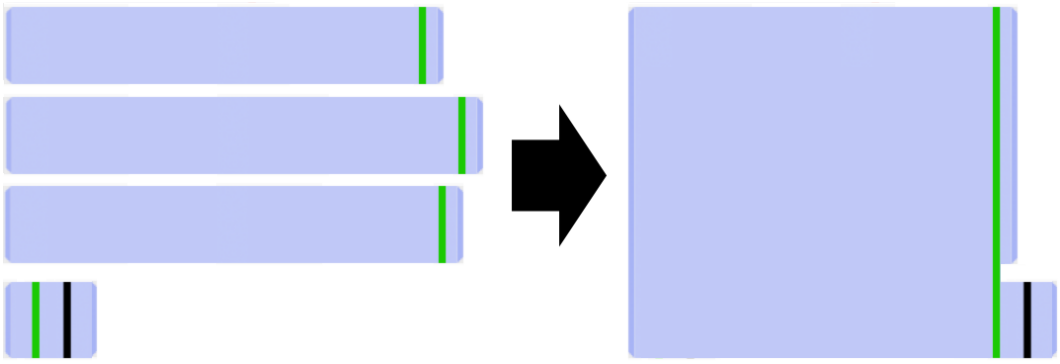


Figure 7.10: EventFlow places the green bar by averaging the time lapse across all records between the start of the record and the green event. The black event then gets placed according to its time lapse relative to the preceding green event. The resulting display obscures the actual time lapse between the start of the record, and the black event.

While this issue was known before my case study with Ms. Young, her study brought to light the potential errors that it can cause during analysis. Ms. Young's case study revolved heavily around comparing the timing of events between two different groups. She was doing much of this comparison using the aggregated view, without delving into the more detailed, time lapse view. Fortunately in her case, the events that she was investigating occurred within one or two events of the alignment point, so the amount that the time lapses were being obscured was minimal. Still though, it became apparent that there was a potential for serious

errors to be made.

To address this problem, the horizontal timeline was removed from the aggregated display. Users are only given a general time scale. This makes it more difficult for users to associate incorrect timing information with a given event. The users now only see explicit times when they scroll over a time lapse to see the detailed view. This view provides correct information about the time lapse between adjacent events. The aggregated display panel was also modified to better highlight the distribution options that can provide users with detailed information about the time lapse between non-adjacent events. This is the intended way for users to access this information, rather than using the former timeline along the x-axis. Figure 7.11 depicts this modification.

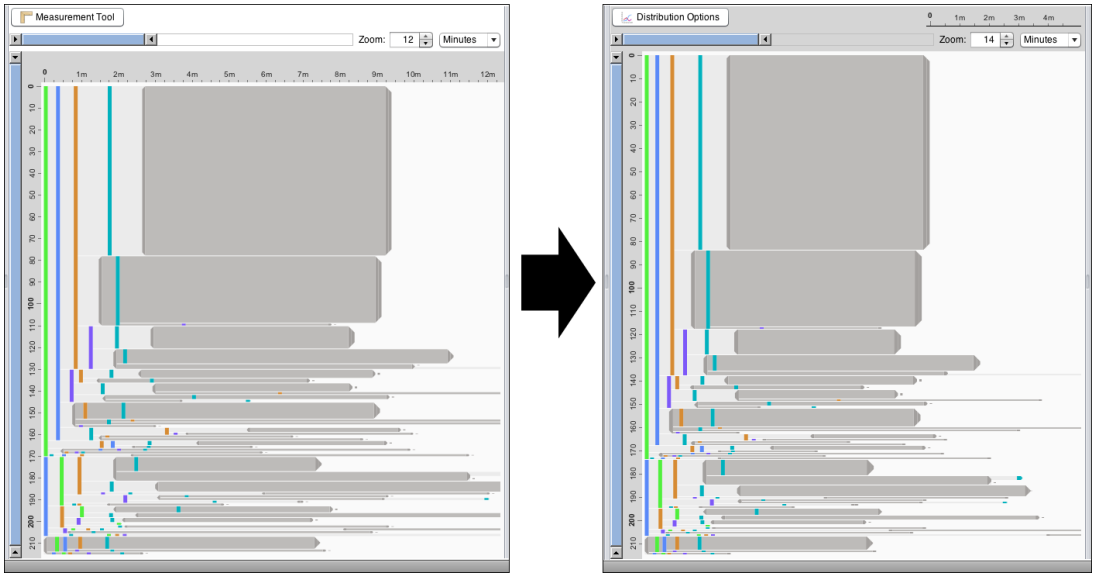


Figure 7.11: The timeline above the aggregated display (left) is replaced with only a general time scale that is removed from the visualization area (right). If users need to assess a time lapse, they must scroll over the space between adjacent events or use the distribution options.

Limitations

Ms. Young's study explicitly highlighted the need to compare groups within EventFlow. Her work revolved entirely around the comparison of event patterns and timing between patients with bone metastasis and patients without bone metastasis. While EventFlow has some mechanisms for comparing groups of records, this is not a forefront feature of the software. Ms. Young accomplished her comparisons by essentially running the application in two windows. Beyond Ms. Young's work, cohort comparison appears to be a pressing need in many real-world research scenarios. A further discussion of how EventFlow could support comparison can be found in Chapter 8.

Summary

Ms. Young was a motivated and self-sufficient user who was able to learn the software and conduct her study with minimal upfront assistance. I saw again, however, that without outside input and feedback, Ms. Young did not take every possible step towards creating a visualization that most clearly conveyed each of her findings. The images that she produced contained extraneous clutter that could have been removed using EventFlow's more precise data transformation tools. She also conducted a series of preprocessing steps (most notably to classify radiation duration) that could have potentially been done using EventFlow. While EventFlow can be a very powerful tool for creating clear images for communication purposes, finding the answers to research objectives is the paramount objective, and one that was clearly met in Ms. Young's study. Furthermore, it is again encouraging to see that results can be achieved without expert knowledge of the software.

7.6 Total Parenteral Nutrition

Total parenteral nutrition (TPN) was introduced in the 1960s, and has become vital in the prevention and reversal of malnutrition for individuals with various

diseases and conditions. TPN is used for patients who cannot or should not get their nutrition through eating. Nutrients such as sugar, carbohydrates, proteins, lipids, and electrolytes are administered intravenously. TPN has been linked to mucosal atrophy, reduced GI hormone secretion, and liver dysfunction. However, the majority of current information on parenteral nutrition-associated liver disease is based on data from the pediatric sector and animal models. Gigi Lipori, the principle investigator for the Integrated Data Repository at the University of Florida, was interested in better understanding the frequency and nature of parenteral nutrition-associated liver disease in adult patients who receive TPN. Her initial dataset is shown in Figure 7.12.

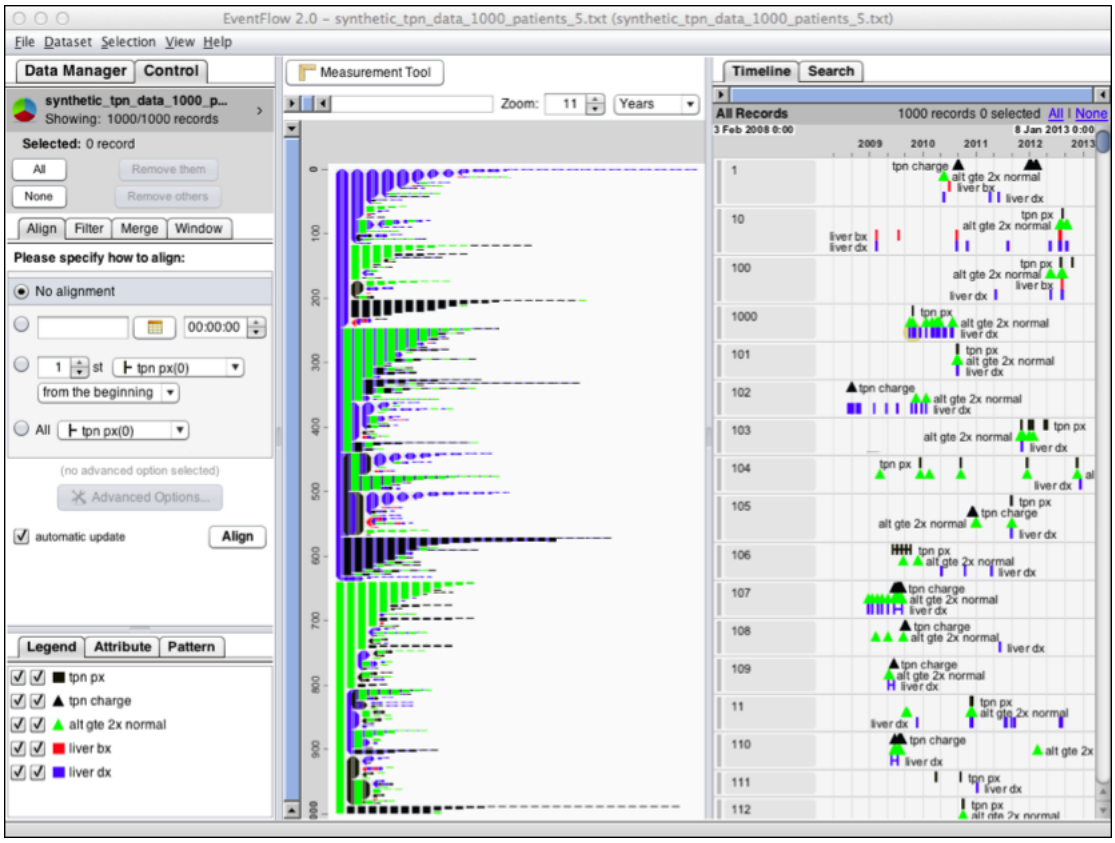


Figure 7.12: Dr. Lipori's initial dataset consisted of two types of TPN events (in black) and three types of liver disease events (in red, green and blue).

Partner: Gigi Lipori

Organization: UF Health

MILCS Level: User-Driven, Mature

Duration: October 2012 - May 2013

Data: A total of 2600 adults who both received TPN and had some indication of liver disease.

Goal: To determine the frequency of parenteral nutrition-associated liver disease, the face of the disease, and the severity with which it presents.

Records: 1000 patients

Average Events per Record: 20 events

Data Cleaning:

1. Replaced TPN Change events (originally a point event) with a derived TPN Change interval event.
2. Merged both TPN events into a single interval category.
3. Merged TPN intervals that were either separated by a 10 day gap or had a 30 day overlap.
4. Created a marker event to indicate the start of each patient's first TPN event.
5. Filtered out remaining TPN events.
6. Aligned dataset to first TPN marker (Figure 7.13).
7. Filtered out patients with liver disease prior to receiving TPN.
8. Created marker event for the first liver disease event in each category (Figure 7.14).

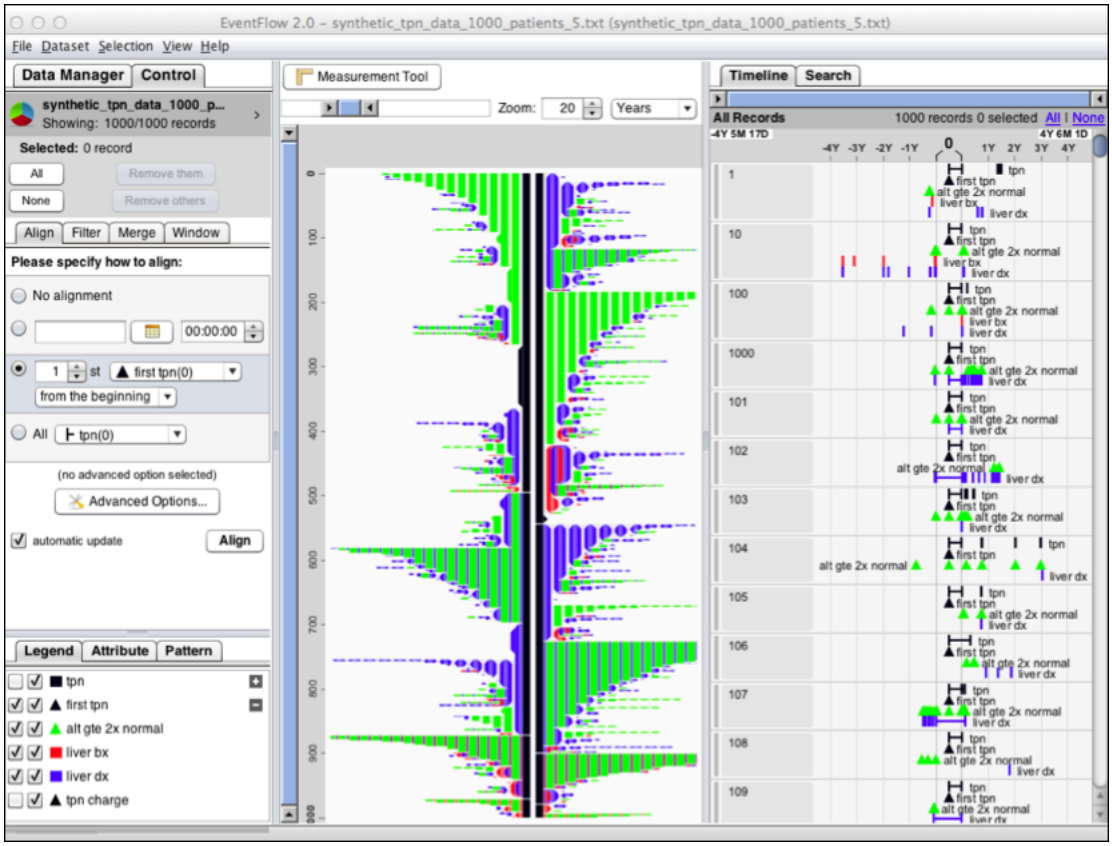


Figure 7.13: Dr. Lipori's transformed dataset, aligned by each patient's first TPN marker event. Patients who did not have liver disease before they received TPN can be seen here without any events to the left of the alignment point (about a quarter of the way down the aggregated display). The dataset was subsequently reduced to only these patients.

Study Procedures

My case study with Dr. Lipori was conducted primarily over e-mail. However, Dr. Lipori was able to meet with me on two occasions during larger user group meetings. She attended these meeting in order to hear about other work that was being done using EventFlow, as well as to present her own findings. Dr. Lipori was the first user to test and make use of EventFlow's Find & Replace system. The majority of my correspondence with Dr. Lipori centered around questions about this interface, and the various data transformations that it was capable of accomplishing. Thus the bulk of our correspondence took place between January

2013 and May 2013, when the Find & Replace system was initially being tested. Dr. Lipori performed multiple transformations and simplifications on the TPN dataset, the results of which can be seen in Figure 7.14.

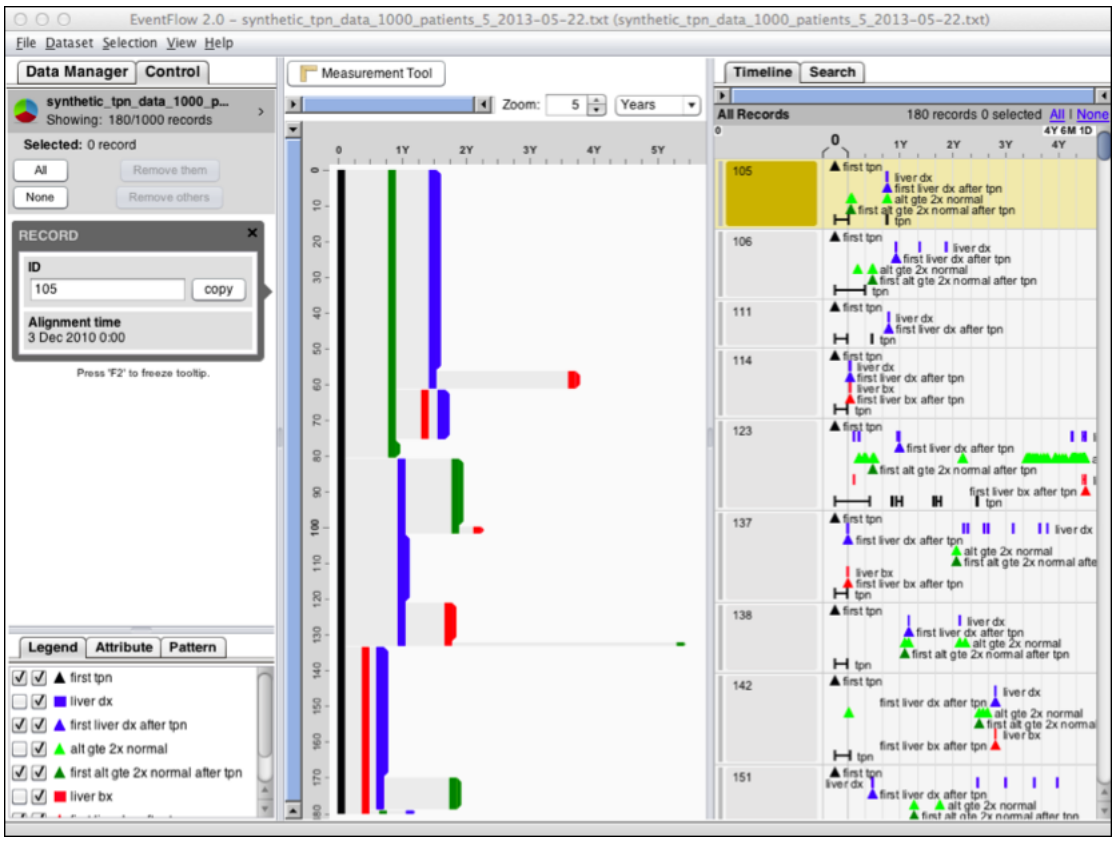


Figure 7.14: Dr. Lipori's final dataset, simplified down to only the patients who experienced liver disease after receiving TPN, and only the first events from each liver disease category.

Outcomes For User

Using EventFlow, Dr. Lipori was able to answer all of her initial research questions about the prevalence and impact of TPN on liver disease. By aligning her dataset by each patient's first TPN event (an event that was derived using Find & Replace, category aggregation, and interval merging), she was able to easily isolate the 180 patients for whom TPN may have triggered the onset of liver disease (see Figure

7.13). These were the patients who had not experienced any liver disease symptoms prior to receiving TPN. From there, Dr. Lipori narrowed her dataset down to the first event in each liver disease category. This allowed her to see both the average onset time of each liver condition after TPN, as well as the distribution of onset times across this subset of patients (see Figures 7.15 and 7.16).

This analysis not only answered Dr. Lipori's initial and most salient research questions, but also prompted her to perform a more open-ended analysis of how liver disease manifests itself in patients who receive TPN. For example, she took a closer look at the progression of alanine aminotransferase (ALT) readings by switching that category with one of its attributes, which defined these values on a more fine grained level (see Figure 7.17). In this exploratory analysis, Dr. Lipori was able to generate new hypotheses that could be tested against her larger dataset. She reported that traditional statistical techniques for evaluating this type of dataset would have required her to have these hypotheses *a priori* in order to properly test them. EventFlow allowed her to generate these new questions simply by exploring her evolving interests.

Outcomes For EventFlow

As mentioned previously, Dr. Lipori was the first user to make extensive use of EventFlow's Find & Replace system. Her work revealed a wide range of bugs in both the interface and the back-end processing of this system. It also motivated a significant number of new features and feature enhancement, including wildcard event matching and point→interval replacements. Dr. Lipori also served as a guide to my other users and case study partners. Her demonstrations at user group meetings and workshops helped to show other users and potential users how EventFlow can be leveraged in real research scenarios.

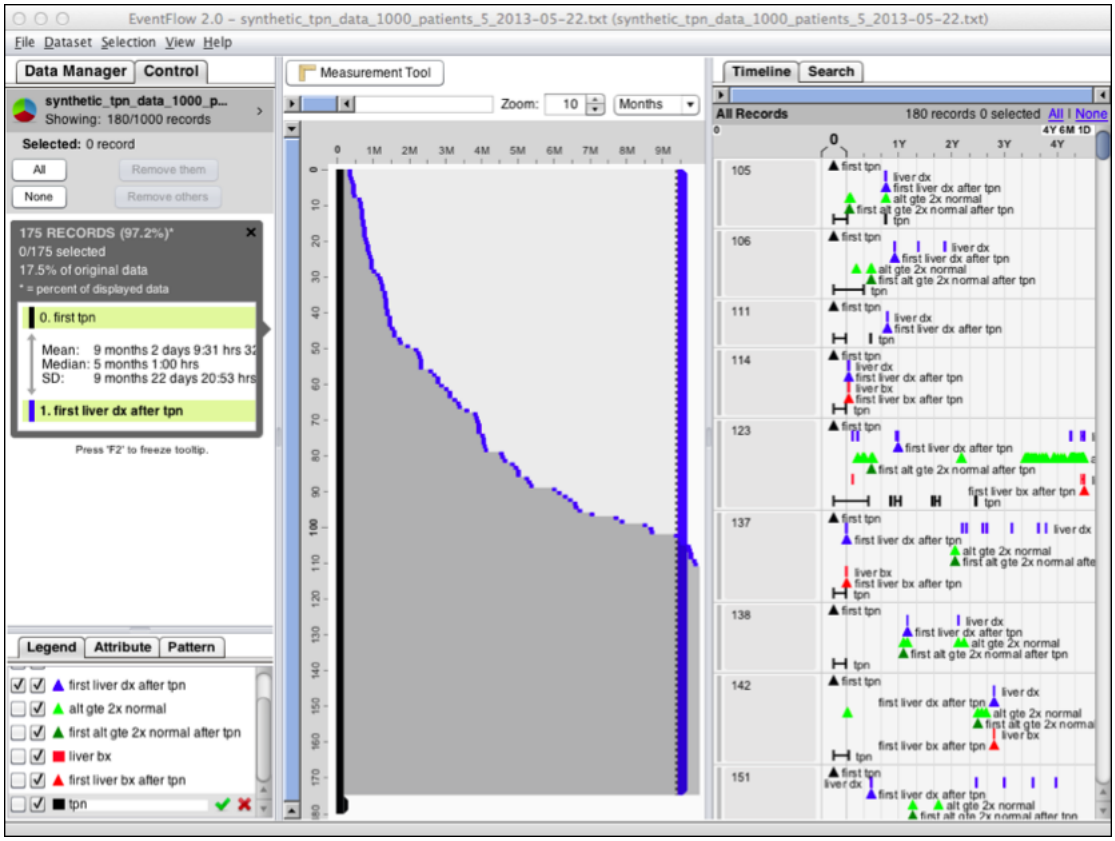


Figure 7.15: By filtering the dataset down to only the patients' first TPN event and first liver disease diagnosis event, Dr. Lipori could use EventFlow's brushing to see the average onset time of this condition.

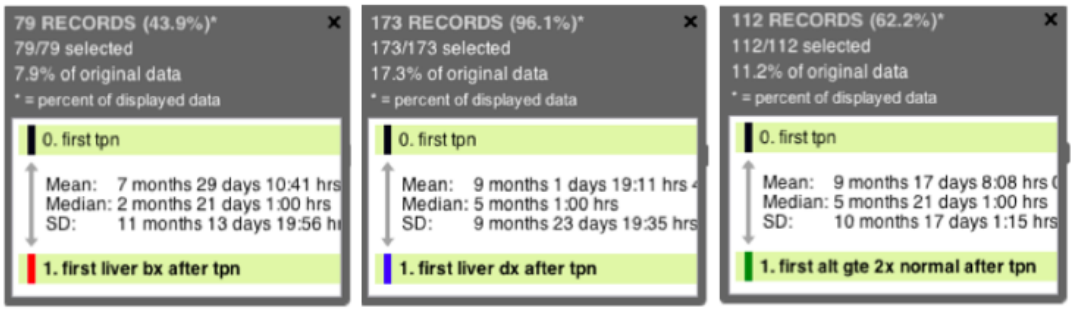


Figure 7.16: Using the same filtering and brushing, Dr. Lipori was able to see the average onset time of each liver condition.

Limitations

Dr. Lipori was not only an engaging case study partner, but also a highly capable data analyst. She was not only able to leverage the existing features of EventFlow,

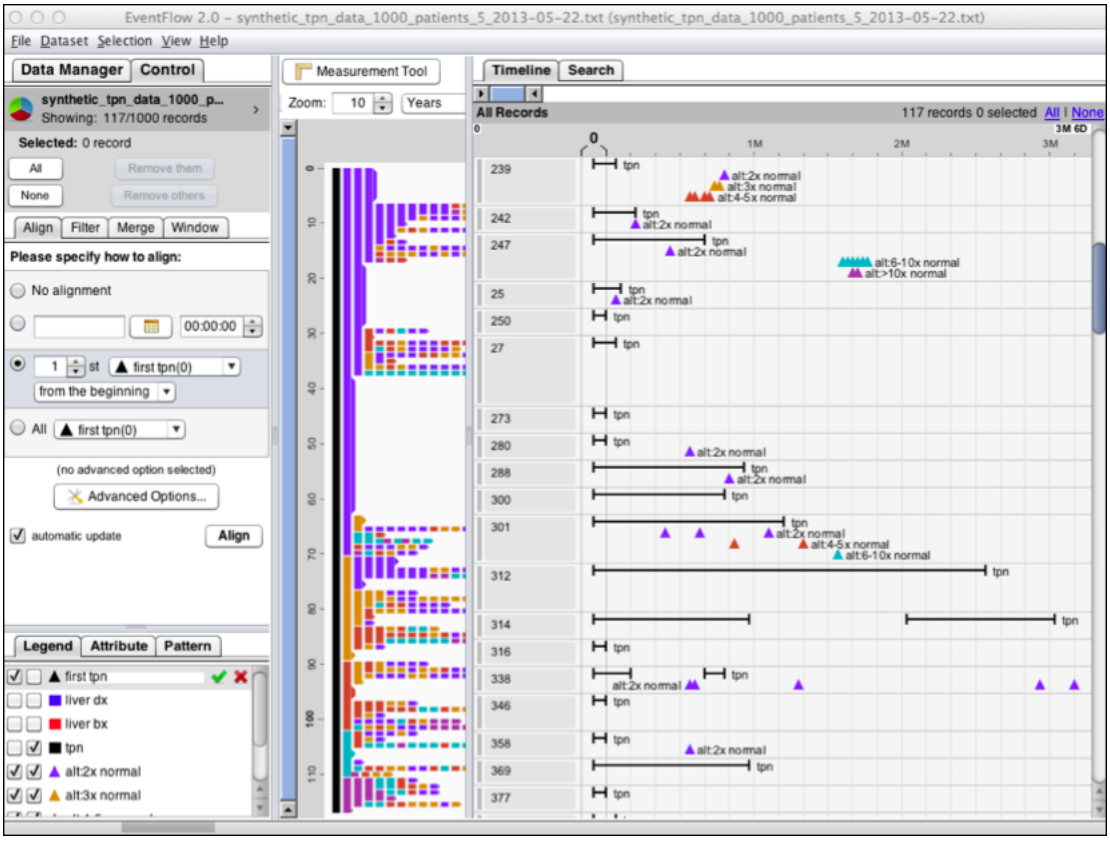


Figure 7.17: Dr. Lipori conducted an exploratory analysis of ALT results at a more fine-grained level.

but was also able to see the bigger picture of my development of the application into a broader tool for data manipulation and analysis. For every question she answered, she generated new questions, bigger questions, and more complex questions. My work with Dr. Lipori illustrated the ultimate need for EventFlow to be integrated into a larger, more comprehensive data analytics tool. As discussed in Section 8.1 of Chapter 8, such a tool would provide support for not only event sequence analysis, but also support for event attribute analysis and metric analysis.

Summary

Dr. Lipori was one of my most engaged and involved case study partners. The total parenteral nutrition analysis was only one of many case studies that she and

I worked on together. Her patience with the development process and confidence in performing more complex data transformations allowed me to develop EventFlow's Find & Replace system from a prototype tool into a fully functioning feature. Dr. Lipori was also instrumental collaborator on bug fixes and feature designs throughout the application.

7.7 Trauma Role and Activity

MedStar Health is a not-for-profit healthcare organization that operates a variety of healthcare-related entities including hospitals, nursing homes, and research facilities. In this case study, I worked with Sarah Parker, a human factors research scientist at MedStar's National Center for Human Factors in Healthcare. Dr. Parker was investigating the various tasks performed by medical staff during a trauma resuscitation. Unlike my case study with Dr. Carter (see Section 7.1), which centered around only a handful of patient-centered events, Dr. Parker was interested in the tasks and interactions between various members of the medical staff. In fact, nearly every communication that occurred during the resuscitation was recorded as an event. Dr. Parker also recorded the stage of the trauma that the event occurred during, and the person who performed the event. In some cases, she also recorded the person who was directly affected by the event. For example, when information was conveyed between trauma team members, she would record who provided the information and to whom. These events were recorded manually by watching videos of each trauma resuscitation. The iPad interface used to enter enter each event is shown in Figure 7.18.

Dr. Parker's research objective was extremely open-ended. She was looking for any interesting pattern of tasks, such as tasks that are typically performed at a certain stage of the trauma or tasks that are typically performed in a certain order. She was also interested in tasks that are performed by a specific role, and how the role of each member of the trauma team might change throughout the

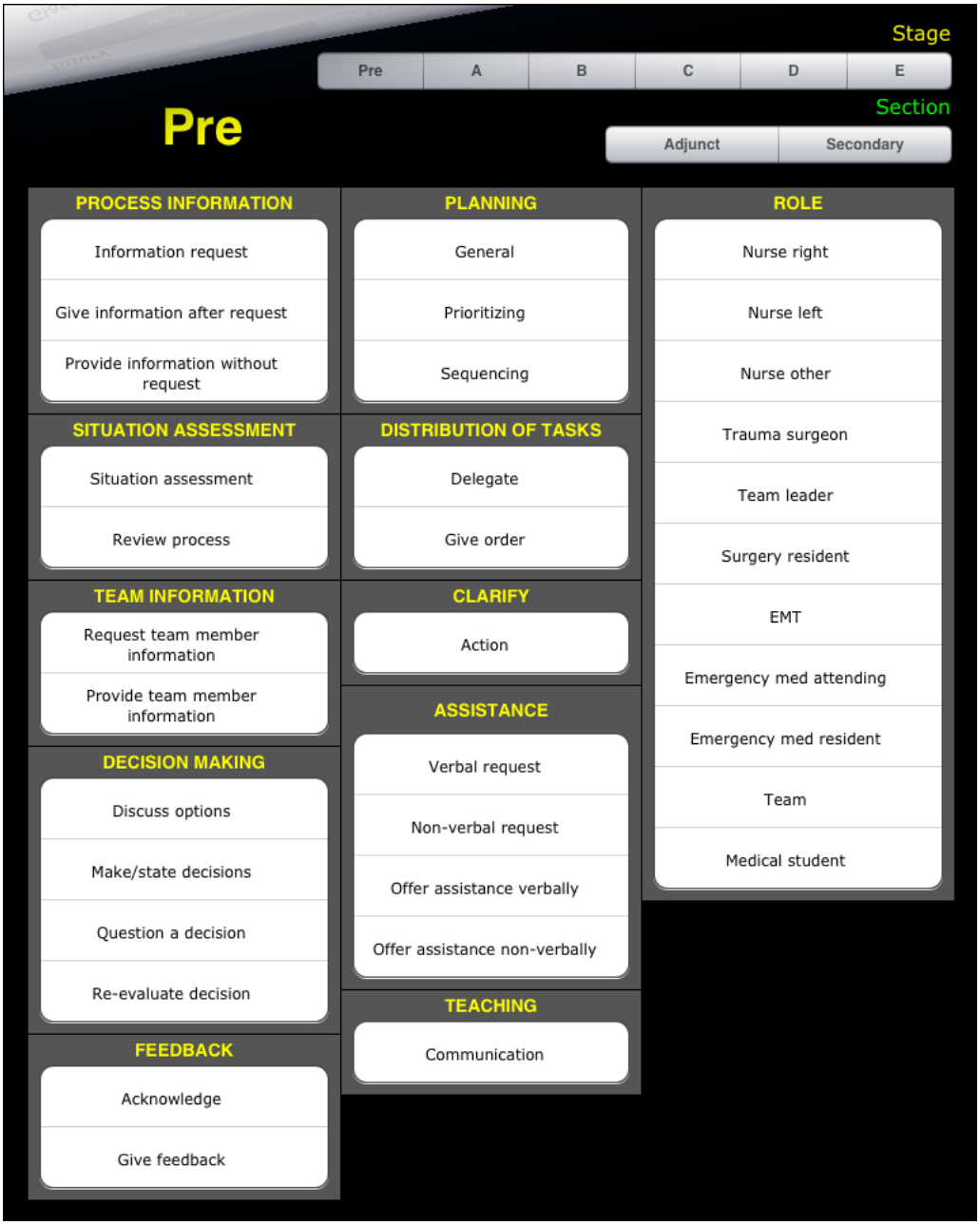


Figure 7.18: The iPad interface that Dr. Parker used to generate her dataset. To input an event, she would first select a stage (at the top), then select the event that took place (left two columns), and finally select the person who performed the event (rightmost column).

resuscitation. Her only initial hypotheses were those that seemed to fit intuition. For example, she expected that team information, such as team member names, to be shared predominantly during the initial "Pre" stage, and that teaching events would frequently involve medical students and residents.

Partner: Sarah Parker

Organization: MedStar Health

MILCS Level: Early, Chauffeur Mode

Duration: February 2013 - November 2013

Data: The tasks performed by various members of the trauma team during a trauma resuscitation.

Goal: To better understand which tasks are typically performed during the various stages of a trauma resuscitation, in what order, and by whom.

Records: 55 patients

Average Events per Record: 51 events

Visual Complexity: 2994 visual elements, 1.93% of display height per element

Study Procedures

I met with Dr. Parker at her MedStar offices on three separate occasions. I also gave her an initial tour of EventFlow via screen sharing. Overall, we conducted four bi-monthly meetings, each lasting about an hour. The final portion of our work together was conducted primarily over e-mail. However, a large component of my work with Dr. Parker involved getting her data into an EventFlow-readable format. Her initial data, shown in Figure 7.19, contained a patient identifier, an event time, and then five columns of relevant information about the event including the stage of the trauma during which the event occurred, the general event category, the

specific event category, the person who performed the event, and the person that the event was directed towards.

record	date/time	stage	category	element	who	to whom
20121026-185538	10/27/12 0:44	Pre	Process Information	Provide information without request	EMT	
20121026-185538	10/27/12 0:45	Pre	Process Information	Provide information without request	Team leader	Team
20121026-185538	10/27/12 0:45	Stage A	Process Information	Provide information without request	Team leader	Team
20121026-185538	10/27/12 0:45	Stage B	Process Information	Provide information without request	Team leader	
20121026-185538	10/27/12 0:46	Stage B	Distribution of Tasks	Give order	Nurse left	Surgery resident
20121026-185538	10/27/12 0:46	Stage B	Process Information	Information request	Trauma surgeon	
20121026-185538	10/27/12 0:46	Stage C	Process Information	Give information after request	Team leader	Trauma surgeon
20121026-185538	10/27/12 0:47	Stage C	Process Information	Information request	Nurse right	Team leader
20121026-185538	10/27/12 0:47	Stage C	Distribution of Tasks	Give order	Trauma surgeon	Team
20121026-185538	10/27/12 0:47	Stage E	Process Information	Provide information without request	Team leader	
20121026-185538	10/27/12 0:48	Secondary	Distribution of Tasks	Give order	Trauma surgeon	
20121026-185538	10/27/12 0:48	Secondary	Clarify	Action	Team leader	
20121026-185538	10/27/12 0:50	Secondary	Assistance	Verbal request	Trauma surgeon	Nurse right
20121026-185538	10/27/12 0:50	Secondary	Feedback	Acknowledge	Nurse right	Trauma surgeon
20121026-185538	10/27/12 0:50	Secondary	Teaching	Communication	Trauma surgeon	Emergency med
20121026-185538	10/27/12 0:50	Secondary	Teaching	Communication	Emergency med	Team leader

Figure 7.19: Dr. Parker's initial data formatting.

The reconfiguring of Dr. Parker's data involved two steps. The first was to covert the last three columns into event attributes. Empty cells (particularly in the "to whom" column) were filled with a place-holder, "(None)" value so that all three of these fields could be accessed through category-attribute switching. Next, the "stage" field was converted into its own interval categories. This way, EventFlow could visualize each event as occurring within a given stage. It would also be possible to filter out all of the point-based categories, to see only the sequence of stages in each trauma. Dr. Parker and I spent about a week conferring over e-mail about these adjustments. The resulting datasets were both conducive to her research objectives and EventFlow-readable.

That is not to say, however, that they were human-readable. Figure 7.20 depicts the initial loading of Dr. Parker's data. With an average of 50 events per record, and nearly 20 unique event categories, the EventFlow display was unable to aggregate records beyond the first four or five events. The overall display was completely unreadable. Because of this, the majority of my sessions with Dr. Parker involved demonstrations of how to look at the dataset piecewise by using EventFlow's simplification features. For example, using the Find & Replace depicted in Figure 7.21, Dr. Parker was able to narrow her data set down to only the Pre stage of each trauma. The resulting dataset is shown in Figure 7.22. From

there, Dr. Parker used EventFlow's overlap filter to determine the prevalence of each event type during this stage. While Dr. Parker expected team information to be exchanged heavily during this stage, it turned out that this occurred in only about half of the resuscitations.

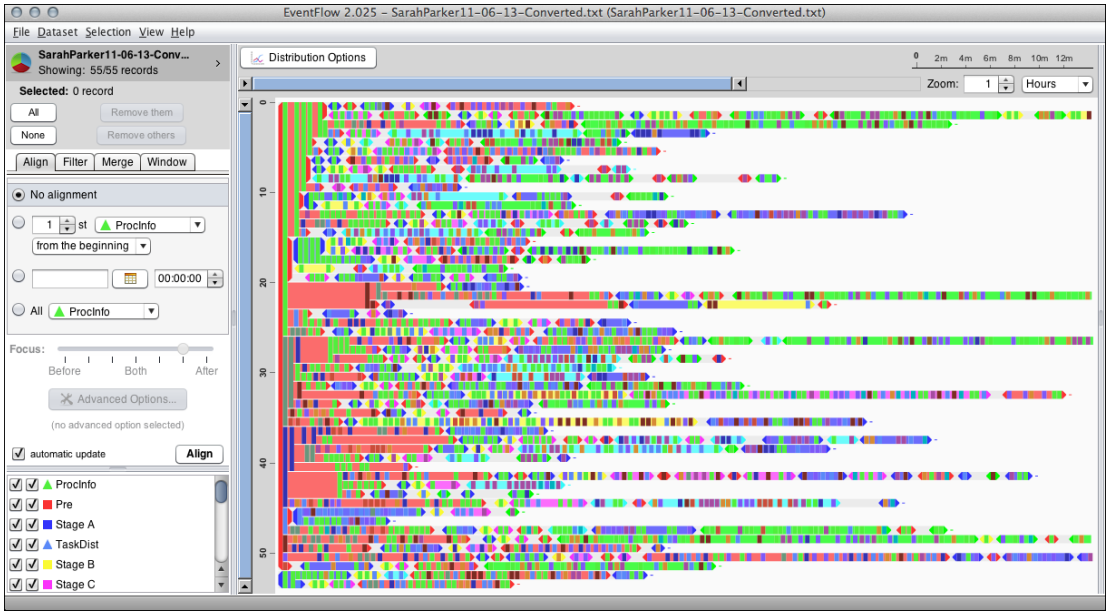


Figure 7.20: The initial loading of Dr. Parker's dataset.

As mentioned previously, Dr. Parker was also able to filter out all of the point-based events in the dataset to see only the sequence of stages in each resuscitation. Figure 7.23 depicts this simplification. Here, Dr. Parker observed that about half of the resuscitations progressed as expected, starting with the Pre stage and then moving through stages A through C. From there, the sequences of stages begins to diverge more visibly. There was also a set of records where the trauma appeared to start over from the beginning midway through. I showed Dr. Parker how to perform various alignments to better see the progress into and out of each stage, as well as how to incrementally insert the point-based events back into the dataset to see how they affected each stage.

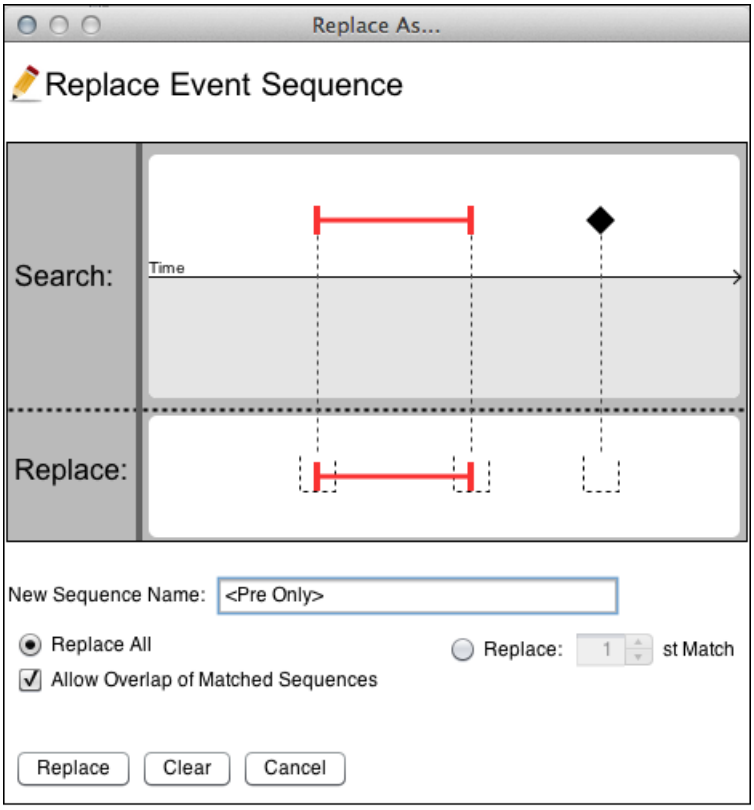


Figure 7.21: Using Find & Replace, all stages other than Pre can be quickly removed from the display. A second query was then performed to filter out records that did not contain a single Pre event.

Outcomes For User

Given that Dr. Parker's objective was so open-ended, and that her dataset was so dense with information, she was able to quickly generate a large number of small observations. She began her study with essentially no knowledge or expectations about what she might find, and as such, nearly every manipulation of the dataset produced some new or interesting finding. Dr. Parker also became much more comfortable with performing various transformations on the dataset. By my last meeting with Dr. Parker, she was able to generate simplification strategies on her own as she came up with new questions about the data. At the time of this writing, Dr. Parker was still in the process of exploring her data and compiling results. It

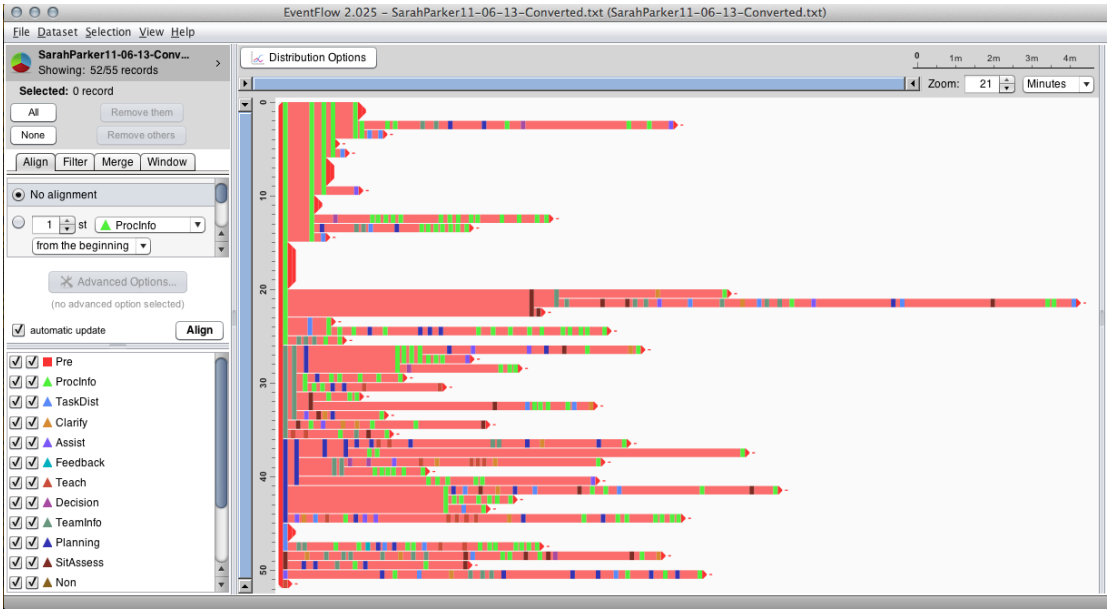


Figure 7.22: Dr. Parker's dataset, narrowed down to only the Pre stage.

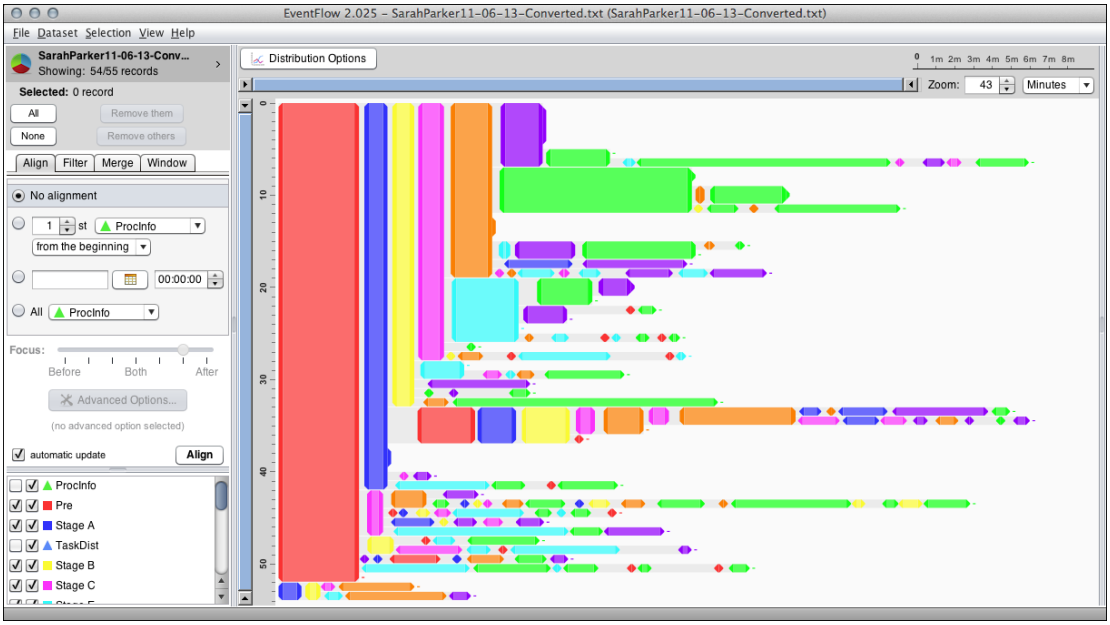


Figure 7.23: Dr. Parker's dataset, narrowed down to only the sequence of stages.

remains to be seen how some of her individual findings might be integrated into an overarching understanding of the trauma resuscitation process.

Outcomes For EventFlow

This case study had the most open-ended objectives of any that I conducted. As such, I was able to see the importance of quickly being able to perform a set of simplifications and then immediately back out those changes to pursue a different question. This study allowed me to better refine the process and interface for saving and re-executing a set of modifications to a dataset. This functionality had previously been tested in situations where only a single set of modifications needed to be performed on a dataset. Because of this, modification files had been designed to have a one-to-one relationship with datasets. Dr. Parker's rapid iteration through different modification strategies, however, provided a completely different use case. Instead of having one modification file associated with a dataset, which would automatically run when the dataset was loaded, I remodeled this feature in order to give users more control of the automatic modification process. Users can now select any modification to be run against their dataset.

Dr. Parker's case study was also the first of many to demonstrate the effect that time granularity can have on EventFlow's individual record display. In Dr. Parker's dataset, the events within a given record typically occurred within a single hour of each other. However, a date was included in the timestamp for every event. Because of this, the temporal range of the dataset vastly exceeded the temporal range of any individual record. This caused the individual display of each record to be significantly compacted (see Figure 7.25). To mitigate this effect, I preprocessed the dataset to normalize each event time against the first event in the record. This change had no impact on Dr. Parker's evaluation since her focus was on relative event timing and sequences within each record. It is also worth noting that this issue only occurs when a dataset is not aligned, as alignment essentially performs the same normalizing operation in relation to the alignment event. While the event time normalization was done outside of EventFlow in this case study, it could also be done using Find & Replace. To do this, a marker event would be inserted at

the timestamp of the first event in each record (Figure 7.24). The dataset could then be aligned by this marker event for the duration of the exploration.

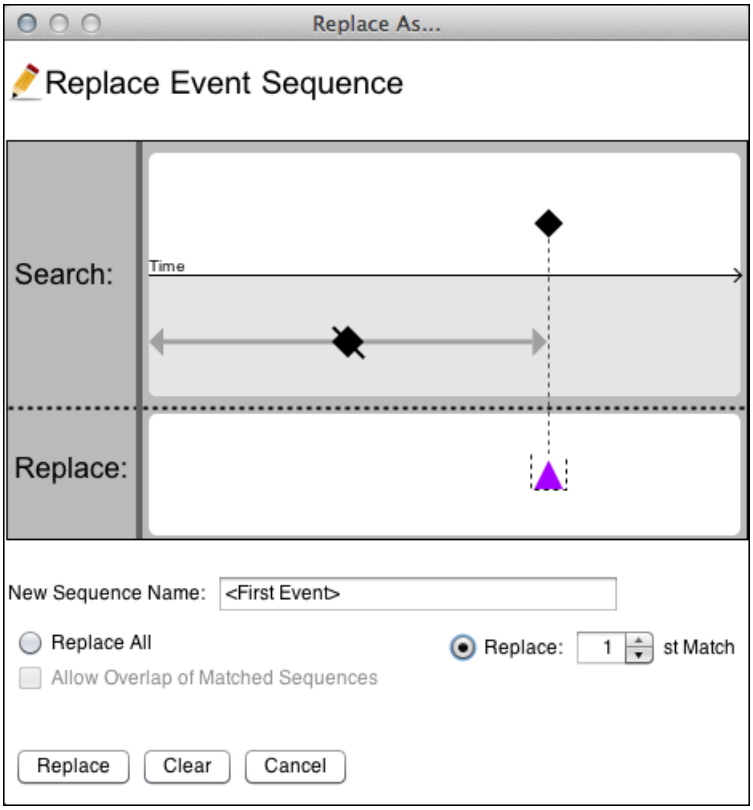


Figure 7.24: A marker event inserted at the first event in each record.

Limitations

This case study highlighted the fact that, when domain expertise is limited, EventFlow can be extremely effective at analyzing complex datasets or answering open-ended research questions, but not necessarily both. The majority of Dr. Parker's insights were derived from drastic reductions in the data complexity that allowed her to focus in on particular event pattern or interaction, which allowed her to more effectively generate hypotheses within a more bounded problem space. While these insights were indeed valuable to Dr. Parker, it was not obvious that they could be pieced together into a more overarching insight about the dataset as a whole.

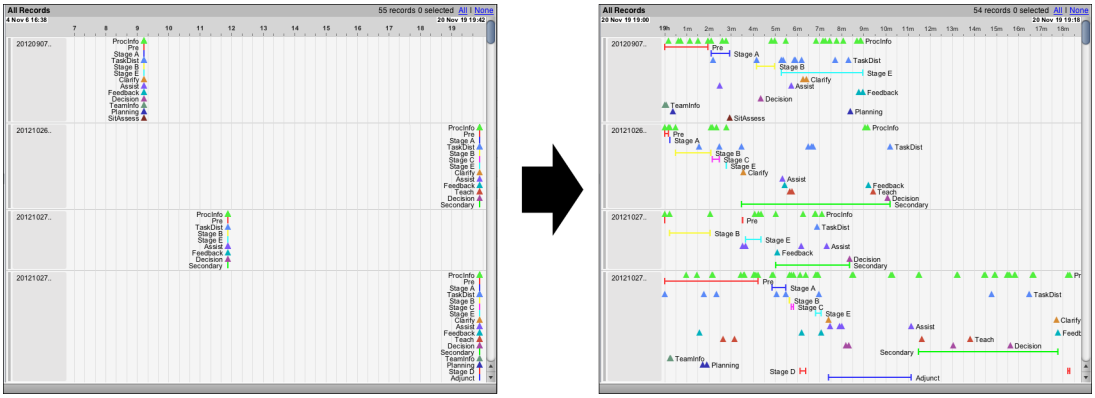


Figure 7.25: The records in Dr. Parker's original dataset (left), are shifted such that the first event in each record is set to a time of zero, and subsequent event times are normalized according to that start time. The result produces a much more readable, individual record display (right).

Summary

The trauma role and activity analysis case study was unique in that it involved a great number of very straightforward simplifications to the initial dataset. Despite this, the initial dataset was so overwhelmingly complex, that it is unlikely that Dr. Parker would have been able to effectively explore the dataset without my assistance and prompting. This suggests that, while EventFlow offers a wide range of effective data transformation solutions, users who are operating without support would be greatly helped by a process model that could guide them through their initial explorations. Dr. Parker became far more self-sufficient and self-motivated once she grew familiar with the types of operations that could be performed..

7.8 Debugging Distributed Storage Systems

VASTech is a South Africa-based software engineering company that works with large computer clusters. Their products rely on a distributed data storage system, which maintains data that is collected from passive network surveillance. My case study partner, John Gilmore, was interested in using EventFlow to better understand the behavior of this storage system. He wanted to be able to characterize

the behavior of the system under normal conditions, as well as detect patterns that occur after failures. His dataset consisted of the message transmissions between nodes in the system, as well as state changes that may have occurred within a single node.

The data storage system stored data as blocks, each of which possessed a unique ID. For every given block, Mr. Gilmore was interested in seeing that block's life cycle in the the system, from the time it was created to the time it was destroyed. For example, in order to ensure that data is not lost when when a server goes down, blocks that currently exist in only one location are replicated to another node. If that node's storage system is full, any new block overwrites the oldest block that was previously being stored at that node. This process translates to the following set of events:

- **AddLocal:** Indicates that the block was stored at a given node. The address of the node on which it was stored is included as an event attribute.
- **RemoveLocal:** Indicates that the block was overwritten at a given node by a newer block. This does not mean that the block has been removed from the system entirely. There could still be a copy at another node due to replication.
- **WriteReplica:** Indicates that the block was replicated to a peer node. Again, the address of the node to which it was replicated is included as an event attribute. A WriteReplica event should be immediately followed by an AddLocal event as the new node writes the replicated block to its storage system.
- **TooOld:** Sometimes, when a block is replicated to a new node, it is already older than any of the existing blocks on that target machine. In this case, that block will not be inserted and a TooOld event will be generated.

Mr. Gilmore's dataset tracked 3792 blocks of data within his distributed storage system (a data block here is equivalent to a record). His initial dataset can be seen in Figure 7.26.

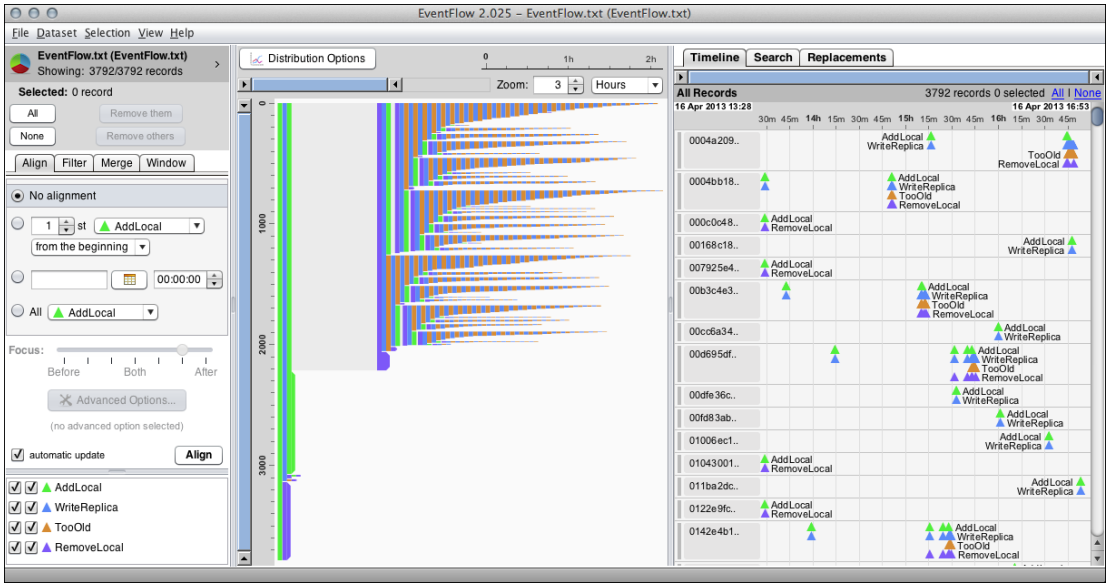


Figure 7.26: Mr. Gilmore's initial dataset of 3792 blocks.

Partner: John Gilmore

Organization: VASTech

MILCS Level: Mature, User-Driven

Duration: March 2013 - June 2013

Data: The lifecycle of data blocks within a distributed storage system.

Goal: To better characterize the behavior of the system both during normal operation and after a crash.

Records at start: 3792 blocks

Average Events per Record at Start: 22 events

Visual Complexity at Start: 7474 visual elements, .3% of display height per

element

Records at end: 2281 blocks

Average Events per Record at End: 9 events

Visual Complexity at End: 366 visual elements, 3.67% of display height per element

Data Cleaning:

1. Replaced WriteReplica→TooOld pairs with a new Replica-TooOld interval (Figure 7.27-Left).
2. Merged adjacent Replica-TooOld intervals (Figure 7.27-Right).
3. Replaced WriteReplica→AddLocal pairs with a new Replica-Add interval (Figure 7.28.
4. Removed the large portion of blocks that were either added and then immediately removed or added and successfully replicated once.

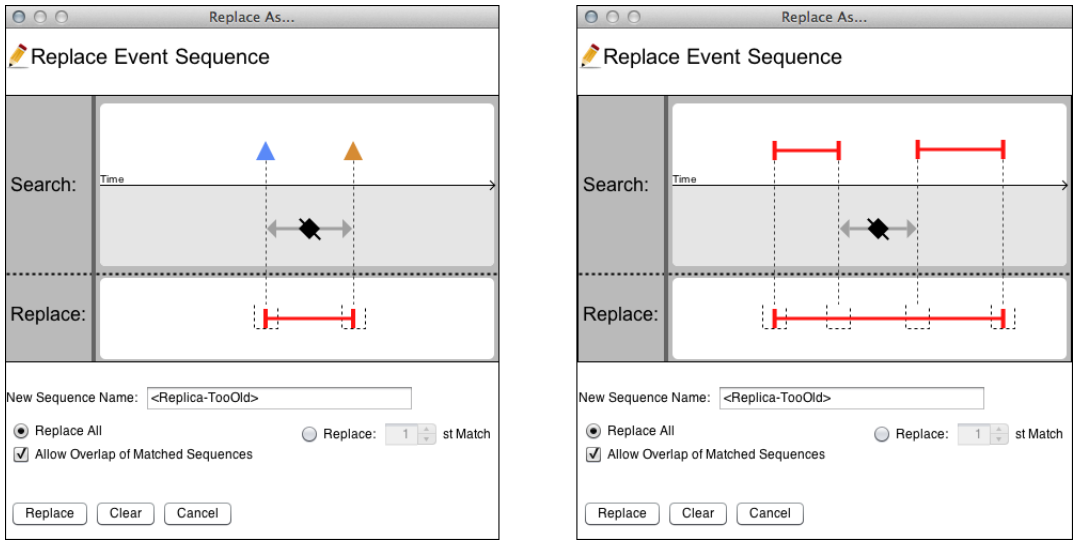


Figure 7.27: Left - WriteReplica→TooOld pairs are replaced with a new Replica-TooOld interval. Right - Adjacent Replica-TooOld intervals are merged.

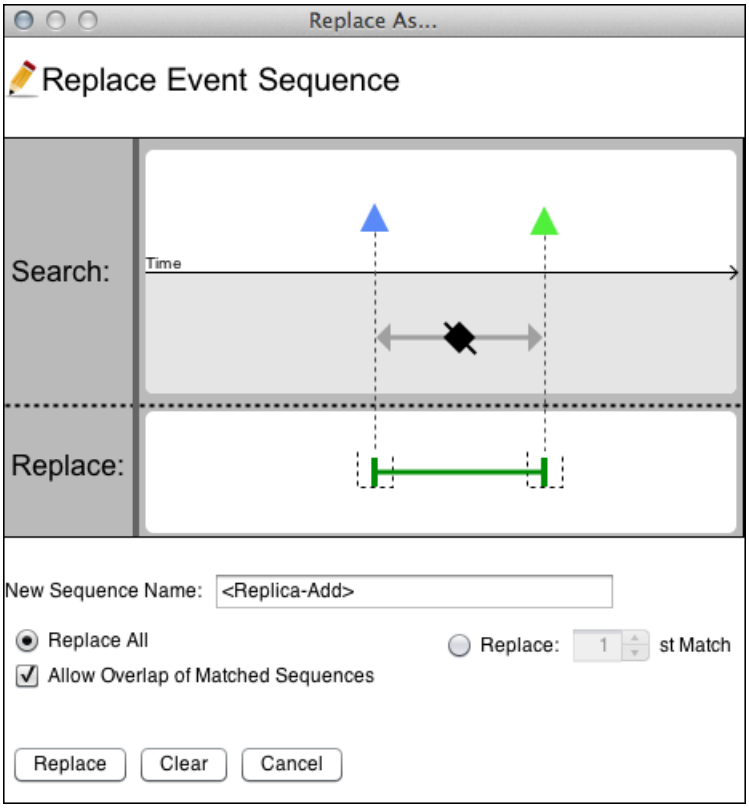


Figure 7.28: WriteReplica→AddLocal pairs are replaced with a new Replica-Add interval.

Study Procedures

The entirety of my case study with Mr. Gilmore was conducted over e-mail due to prohibitively large time zone differences. Across four months, Mr. Gilmore and I exchanged approximately 40 e-mails. Mr. Gilmore was able to get his datasets up and running with minimal help. The majority of our e-mails centered around his questions about specific data transformations, feature requests, and bug reporting. As a programmer, Mr. Gilmore needed only basic guidance in constructing effective data transformations. For example, when I suggested that the frequently occurring WriteReplica→TooOld pairs be replaced with a single interval, Mr. Gilmore was able to deduce on his own that the key to this replacement was to also specify the absence of other event types between these two events.

This case study was also the only study to involve a dataset that heavily revolved around milliseconds of time. While EventFlow supports time granularity down to the millisecond, this is mainly a consequence of the convenient, `long` data type in Java. In most cases, this level of granularity had only been used to artificially impose an ordering between events when none existed. By the end of our four month study, I had given Mr. Gilmore a custom version of EventFlow that displayed all timestamps down to the millisecond. This helped him to judge time lapses and time differences more accurately throughout the remainder of his study.

Outcomes For User

Mr. Gilmore's goal was to be able to better characterize the behavior of his distributed storage system, as well as spot individual and general errors in the way that these blocks of data are operated on. A given data block is used by many different services during its lifetime, and no one service knows what other services have operated on that block. However, to verify that the system is working correctly on a holistic level, he wanted to ensure that services were operating on records in the correct order. This process had previously been invisible to him, and he reported that EventFlow was extremely helpful in elucidating this process. Mr. Gilmore's final, simplified dataset can be found in Figure 7.29.

Fortunately, Mr. Gilmore did not find any overarching errors in the way that his system operated. However, he reported that being able to see this using EventFlow allowed him to eliminate any doubts. He reported that EventFlow helped both him and his team communicate about the behavior of their system, and that it gave them a language by which to describe both normal operation and the anomalous event sequences that occur after a crash. At the end of my study with Mr. Gilmore, use of EventFlow was increasing through the company, and he was in the process of refining his event capture framework in order to squeeze more information into his datasets.

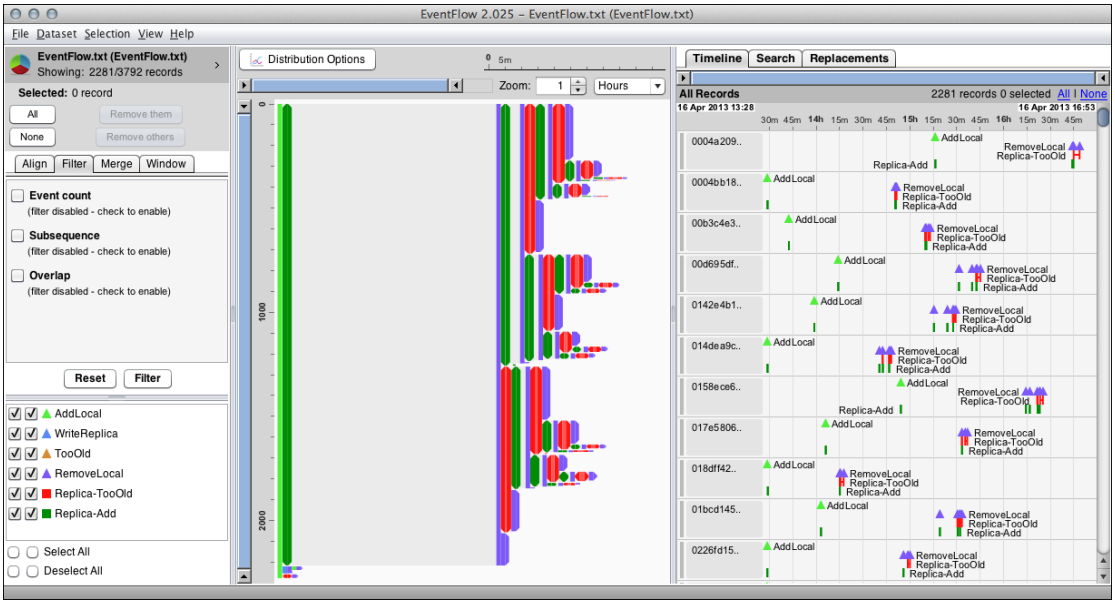


Figure 7.29: Mr. Gilmore's simplified dataset of 2281 blocks.

Outcomes For EventFlow

Mr. Gilmore was very interested in anomalous patterns in his dataset, as these were most indicative of a malfunctioning at some level. However, with 3792 records and 963 unique sequences in the initial dataset, patterns that occurred in only a single record were not large enough to appear in the display. The problem was that it was not obvious to user when this was occurring. There was no easy way to tell whether the aggregated display was showing every sequence in the dataset or hiding certain, infrequently occurring sequences. Mr. Gilmore was only able to notice these hidden sequencesf because the vertical scale in the aggregate display, which should list the total number of records, was listing only a partial record count.

The reality of the EventFlow display is that when the number of unique sequences in a dataset exceeds the number of vertical pixels on the screen, it is not possible to display every sequence. However, it is critical that users are aware of when and where this is happening. To mitigate this issue, EventFlow's aggre-

gated display was augmented to include an alert message that appears when not every sequence can be displayed (see Figure 7.30). When this happens, users can zoom into a particular area of the display, or remove larger chunks of uninteresting records in order to see outlier sequences.

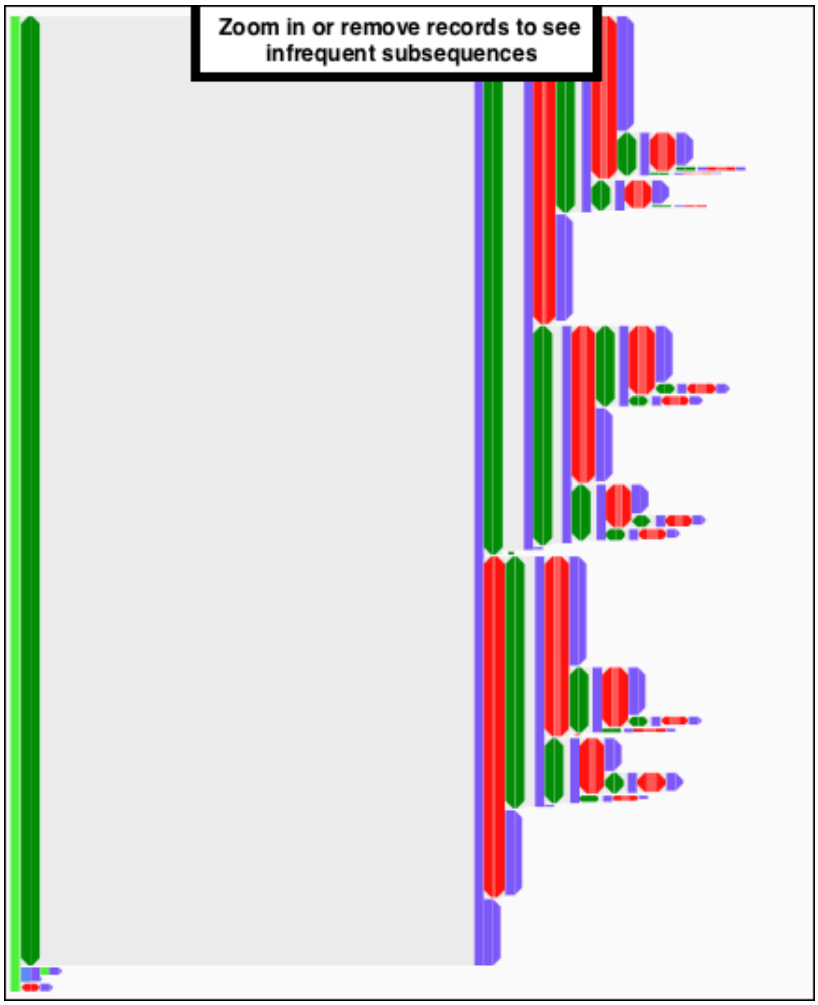


Figure 7.30: EventFlow was updated to alert users when sequences are too infrequent to be displayed in the aggregated view.

Limitations

My case study with Mr. Gilmore was the only one that did not involve any synchronous communication due to restrictove time differences. This hindered my

ability to stay in sync with his evolving objectives, and to understand the finer details of his analyses. Ultimately, this made it difficult to suggest data transformations and simplifications beyond the features that were clearly generating visual clutter. While this lack of synchronicity did not prevent Mr. Gilmore from reaching his intended insights, it may have limited his ability to generate insights beyond the initial research objectives.

Summary

My case study with Mr. Gilmore served to illustrate the utility of EventFlow on all temporal event datasets, not just those in the medical domain. It also demonstrated that a person with more programming expertise, such as Mr. Gilmore, could be much more self-sufficient and confident in designing and executing data transformations. Mr. Gilmore was able to extensively simplify his dataset with only minimal instruction. He was also better able to distinguish between bugs in the software and errors in his own query specifications.

This study also increased my awareness of time granularity and the affect it can have on the overall application. When datasets are highly dependent on milliseconds of time, there is far less breathing room between events, due to the data type that EventFlow uses to encode timestamps. Operations that had been standard for other datasets, such as incrementing an interval end point by one millisecond when the start and end points occurred at the same time, had to be performed with more consideration at this level of granularity.

7.9 Summary

This chapter reports on eight case studies, conducted with real users and real research objectives. These case studies were chosen to highlight the primary contribution of this thesis: the ability to transform an initially complex dataset into a representation that allows the users' research objectives to be addressed. In each

of these cases, visual exploration was not sufficient in allowing users to reach their intended insights. Initial datasets either contained an excessive amount of unnecessary clutter or did not accurately represent real world events or the goal of the study. Through precise querying and a series of data transformation techniques, users were able to generate novel insights and intelligible, informative displays that could be use to communicate those insights to a broader audience.

Chapter 8

Conclusion and Future Directions

Temporal event data has emerged as an integral data type in the field of predictive analytics. The difficulty of this data type, however, is that even though it is particularly well-suited to standard relational storage structures, it is not well-suited to standard relational query specification and processing. Because of this, users frequently interact with this data in ways that do not match their natural way of conceptualizing and understanding event patterns. There is a need among both expert and non-expert users for visualization and graphic-based interaction tools that facilitate the process of analyzing and manipulating temporal event data. My work has addressed this need with four primary contributions:

- **A novel interface for specifying queries over points, intervals, and absences.** The integration of interval-based events and absences into a temporal query system vastly increases not only the inherent complexity of the queries that can be specified, but also the number of event configurations that must be specified in order to capture general temporal concepts such as "during" and "overlaps." It is extremely tempting to focus on integrating this complexity into the query interface alone. However, this can overcomplicate the interface, and in reality, query errors frequently stem, not from the interface, but from the users' initial expectations about a dataset as well as their inherent understanding of temporal relationships. Because of this, my focus was not only on the design of a usable query interface, but also on its integration with a result display that facilitates error detection and rapid query reformulation. The result is an overarching system that allows users to better understand the nuances of temporal relationships in their dataset

as they narrow in on their intended result set.

- **An integer programming strategy for processing temporal queries.** The strategies used to process temporal event queries have an impact not only on computational metrics such as time and space complexity, but also on user-facing issues such as best-match determination and result ranking. An integer programming back-end offers tangible advantages over standard approaches such as within-record and between-record ranking via the objective function, independence from record event ordering, and independent query constraint specification. I demonstrate the feasibility of this approach, not only from a query scope perspective, but also from a scalability perspective.

- **A Find & Replace system for transforming event sequences.** Many visualization and visual analytics systems are built with an ivory tower mindset, where data is clean and interpretable and need only be displayed in order for insights to tumble out. In reality, temporal event data comes from a variety of sources, each with its own biases and motivations that can significantly impact the way that data is both recorded and stored. Researchers must clean data that they did not collect, analyze data that they did not clean, utilize tools that they did not build, and yet still, at the end of it all, convey insight that preserves a logical (and accurate AND reproducible) continuity. EventFlow's Find & Replace system allows users to visually engage with their data at an earlier stage of the analysis process in order to produce datasets that more accurately reflect their objectives and intentions. As a result, they are able to better explore and communicate about trends and findings that were previously obscured.

- **Eight case studies that demonstrate the utility and validity of these approaches.** By working with real word domain experts, I was able to observe both the limitations of EventFlow's initial implementation, as well

as the impact that effective query and data transformation tools can have on the analysis process. Over the course of this work, I moved from analyses that could not even get off the ground due to complications in the underlying data, to collaborations that could be followed through to the intended insight.

Overall, this work is intended to provide a practical link between collecting data and deriving insight through visual analysis. This is accomplished by targeting the intermediate data-wrangling stage of the research process. Data cleaning and data wrangling efforts are not only time and labor intensive, but can substantially dictate the success of the eventual analysis. Providing users with better visual cues and feedback during this stage is a pressing need that has largely been ignored. EventFlow, a tool previously designed only for visual exploration, can now be utilized across a much wider spectrum of the data analysis process to more accurately align datasets to the questions and objectives at hand.

8.1 Limitations of EventFlow

Despite the power of visual analysis, and the contributions of this work, EventFlow still has limitations. In Section 8.2, I discuss potential avenues of future work, which propose extensions to the current implementation. However, the limitations discussed here would likely be addressed outside of EventFlow, either in a separate implementation or as an integration into a larger piece of software.

8.1.1 Domain Expertise

The event-level transformations and simplifications discussed in this dissertation rely on the domain expertise of the users performing those operations. It is not realistic to expect an average asthma-sufferer to be able to transform the LABA dataset with the same precision as an expert epidemiologist. This means that the insights that are revealed as a result of performing those transformations are not

accessible to the average user. When an initial dataset is too noisy to discern meaningful trends, non-expert users could potentially benefit from automated assistance in selecting and performing helpful transformations.

8.1.2 Metric and Attribute Analysis

EventFlow focuses on event patterns and sequences. Metric considerations, such as the day of the week when an event occurred, and attribute considerations, such as the player who took a shot in a basketball game, are not addressed with the same level of detail or accessibility. Most notably, EventFlow's aggregated display explicitly masks metric relationships between events in order to better support sequence analysis. While this prioritizing is intentional, EventFlow is still limited in its ability to answer metric and attribute-based questions. By contrast, other temporal event analytics tools focus heavily on these aspects, but do not provide comprehensive support at the sequence level. The ideal tool for temporal event sequence analysis would provide comprehensive support across event sequence, metric, and attribute considerations.

8.1.3 Over-Transformation

As mentioned in Chapter 6, the ability to transform temporal event sequences also comes with the danger of creating inaccurate and misleading results. While EventFlow is equipped with various mechanisms for preventing this from happening unintentionally, it does not prevent users from intentionally creating misleading displays. A commercial implementation of this software could provide better security by integrating datasets with their provenance in a way that could not be decoupled.

8.1.4 Few Records, Many Events

Ultimately, EventFlow is designed to scale well against datasets with a large number of records. It scales less gracefully as the number of events in each record increases. As a result, datasets consisting of less than 100 records, but more than 40 events per record, are not well-suited to the type of analysis that EventFlow supports. As event sequences get longer, the need for more automated, statistical analysis becomes more pronounced. The complexities of event relationships at this point exceed the capabilities of natural human reasoning, even with visual support.

8.2 Future Work

While this thesis contributes significant extensions to EventFlow's capabilities, as well as a better understanding of the role that visualization can play in the larger context of data analysis, there is still a huge opportunity to leverage this tool to support user needs. In this section I discuss various ways in which this work can be further developed.

8.2.1 Cohort Comparison

EventFlow currently allows users to segment the aggregated display based on a predetermined, record-level attribute. It is also possible for users to create a record-level attribute within the application (via query and selection) in order to produce this segmentation (see Figure 8.1). The goal of this feature is to allow users to visually compare multiple groups of records. This relatively simple feature however, is dwarfed by both the prevalence and the complexity of the need to compare two or more groups of records. Many of the case study partners that I worked with have conducted some form of this comparison as either a central or peripheral component of their research. In many cases, this need was addressed by simply running EventFlow concurrently in two different windows.

Given how frequently this need arises, EventFlow could be extended to more explicitly support comparison. For example, in the segmented display, if a user clicks on a particular event sequence in one segment, the visualization could highlight that sequence in the other segments in order for users to better gauge its prevalence in other groups. There are also a whole host of comparison statistics that could be computed at the event level, the record level, and the group level. The various features of the dataset that could be leveraged to compare two groups of records range from the number of unique sequences in each group to the shape of the time lapse distribution between events in a shared sequence.

Unlike the more exploratory questions that EventFlow is currently designed to support, cohort comparison is a much more bounded task. Because of this, it may be possible to automate various components of this comparison. Users could be given a precomputed report on the differences between two groups, and then use the visualization to better gauge why these factors might be important. Custom controls could be developed as well, to facilitate the interactions and manipulations that are unique to this particular branch of the problem space. From an overarching research perspective, cohort comparison has emerged as the most relevant next step for the software to take.

8.2.2 Database Integration

In order to make EventFlow accessible to new users, the entire Java-based application runs in main memory. Users can download a single executable and a sample dataset, and can have the application up and running in only a few clicks. However, this also means that the size of the entire dataset and the resulting data structures is limited to the Java virtual memory stores on each user's machine. Were EventFlow to be implemented in a commercial setting, the application would need to be modified such that the bulk of the underlying data was maintained in a secondary storage system. This implementation would involve considerations at six different levels: basic storage structures, the working dataset, individual display access,

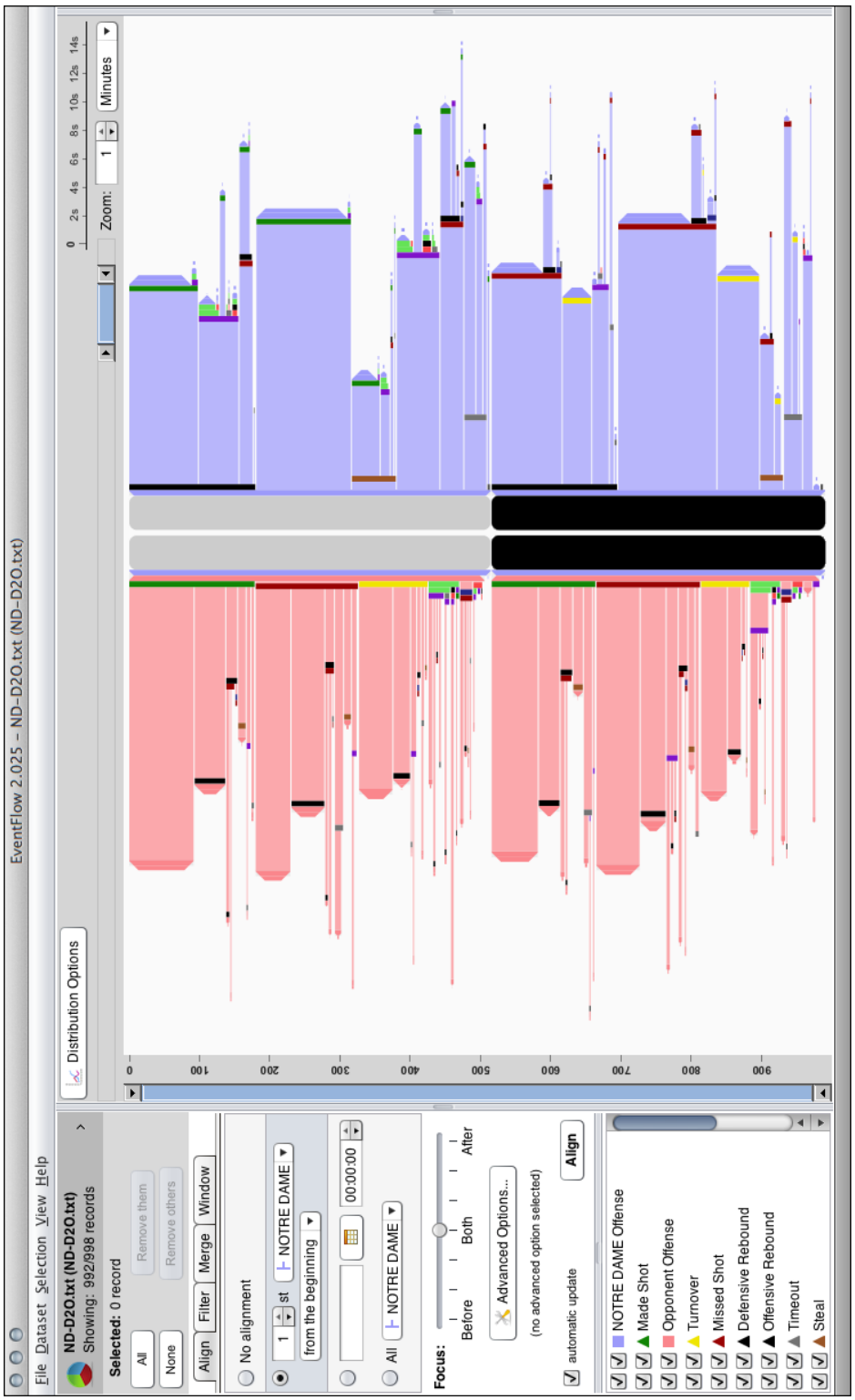


Figure 8.1: A basketball dataset is segmented into possessions where the offense scored points (top) and possessions where the offense did not score points (bottom).

aggregated display access, event sequence query, and event sequence modification storage. In the section, I provide general recommendations at each of these levels.

8.2.2.1 Basic Storage Structures

Instead of a single, text-based input file, a database back-end would allow temporal event datasets to be more cleanly structured into hierarchical database relations. At the top-most level, each dataset would have a unique identifier and a name (Dataset Table):

Dataset ID	Dataset Name

Each dataset would consist of a set of records (Dataset Record Table):

Dataset ID	Record ID

Each record would have a unique identifier and a name. Here, the record name would be value that EventFlow currently uses as the record identifier. That is, the record name would be the externally specified record identifier (Record Table):

Record ID	Record Name

Each record would consist of a set of events (Record Event Table):

Record ID	Event ID

Each event would have a unique identifier and an event type. The event type would indicate whether the event was the start of an interval, the end of an interval, or a point event. The existence of the start of an interval would imply that there exists the end of an interval with the same event ID. Thus the event ID

and the event type together would form the unique identifier for each event. An additional constraint could also ensure that an interval's start time occurs before it's end time. Each event would also have a category and a timestamp, much like the current EventFlow data format (Event Table):

Event ID	Event Type	Event Category	Event Time

Two additional tables would maintain the record-level and event-level attributes:

Record ID	Attribute Name	Attribute Value

Event ID	Attribute Name	Attribute Value

Note that, on the event level, this structure implies that the start point and end point of an interval share the same set of attributes. While this reflects EventFlow's current setup, it would also be possible to include the event type in the event attribute table to allow interval start and end points to have unique attributes.

8.2.2.2 Working Dataset

The working dataset is the list of records that EventFlow is currently displaying in both the individual and aggregated view. It is also the list of records that is queried over and modified when a Find & Replace operation is performed. Much like the current implementation, EventFlow would maintain the list of record IDs that constitute the working dataset. This list would be passed to the database when the application needs to either execute a query, or recompute the display.

8.2.2.3 Individual Record Display

The back-end implementation of EventFlow's individual record display would arguably be the most straightforward. In this display, records are listed in a vertical,

scrollable interface. The user can hover over any record ID, or any individual event to see its attributes. As such, the record, event, and attribute details for the records currently being displayed would need to be present. However, to minimize the strain on main memory, the application could cache this information only for the two sets of records that are adjacent to the record set that is currently being displayed. This would provide rapid access to this information if the user scrolls up or down without requiring an unnecessarily large portion of the dataset to be kept in main memory.

8.2.2.4 Aggregated Record Display

EventFlow's aggregated data structure (the event tree structure) is likely to be the most memory intensive, even using a database back-end. At the very least, the largest amount of data will be passing through this component of the display. However, the application could be easily modified such that a significantly smaller amount of this data is actually retained by the application. Furthermore, this could likely be done without significantly detracting from the current experience. Here I list some of the primary considerations:

- **Data Preprocessing:** Despite the fact the entire dataset must be processed into the tree structure, the tree is constructed incrementally, one record at a time. This means that only a small cache of event records needs to be maintained by the application at any given time. The database, using a very straightforward query, could deliver each record to EventFlow's preprocessor, already sorted, and containing only the active event categories. If an alignment is in place, the time of the alignment point could be computed using a second query. The EventFlow tree processor only needs these two components, the sorted event record and the alignment time, to build a single record into the tree.

- **Span Structures:** As discussed in Chapter 3, Section 3.2.2, EventFlow

maintains a list of every open interval between adjacent pairs of events. This allows the tree structure to distinguish between events that are occurring in isolation, and events that are occurring during another event. This distinction creates different branches in the resulting tree. Currently, when a record is processed into the tree structure, a subset of this list is derived based on the list of active event categories. To large extent, however, this subset is the only component of the span structure that is needed. Instead of constantly maintaining the full list of open intervals, this subset could be computed on the fly, as the tree is being built. The only complication would be handling open intervals around the alignment point. The tree processor, which only handles one side of the alignment at a time, would have to infer which intervals are open at the alignment point based on which intervals were not closed on that side of the alignment. Additional considerations would have to be made for EventFlow's overlap filter (discussed in Chapter 4, Section 4.2). This is the only component of the application that uses the full span structures to detect relevant overlappings. This module would have to be altered, either to support only visible events, which would allow it to search the tree structure rather than the event records, or to locate relevant overlappings by the event end-points rather than using the span structures as a shortcut.

- **Event Node Storage:** Currently, each node of the tree structure maintains a list of the record IDs that are aggregated at that node. This is because EventFlow's internal indexing is done only by record. However, a database back-end could provide indexing by both record IDs and event IDs. This would allow each node in the tree structure to maintains a list of the event IDs, rather than the record IDs. The event indexing could be used to quickly retrieve a list of record IDs based on the list of event IDs, which would provide the interaction between the aggregated record display and the indi-

vidual record display. The event IDs would also provide faster access to the underlying distributions, discussed next.

- **Event Distributions:** Users can currently hover over any time lapse in the aggregated display to see the distribution of individual time lapses that comprise that average. However, this distribution is presented in many cases where it is not explicitly needed and in fact, this feature is frequently turned off when it becomes a distraction. Furthermore, if users need to see a distribution for non-adjacent events, they must explicitly request it. To minimize the amount of memory that is retained within the application, it would make sense to require that users ask to see all time lapse distributions. By storing the event IDs at each node, the database could quickly be queried for the timestamps of the events that comprise the averaged time lapse. This query could also return the corresponding record IDs for faster interaction with the individual record display.

- **Attribute Node Storage:** Integrating attribute nodes in to the aggregated display would be quite simple using a database back-end. When records in the working dataset are queried to be processed into the tree structure, the query would simply be augmented with a GROUP BY clause to segment the records by the specified attributes. The tree processor would use this groups to determine how to create the attribute node of the tree.

- **Attribute Distributions:** Much like the proposed functionality of the event distributions, the attribute distributions are currently accessed by a direct request from the user. With a database back-end, the specified attribute would be queried for using the event IDs at the selected node and passed back to the application.

8.2.2.5 Event Sequence Query

Standard relational query processing is not well-suited to temporal event querying. Without custom processing techniques, these queries can result in a lengthy sequence of self-join operations. The translation of such a query into a standard database language such as SQL is also not a trivial matter. Because of this, it would likely be best for queries to be processed within EventFlow. The database back-end can be used to efficiently retrieve the relevant events in each record, allowing the application to execute the query either individually or in parallel.

8.2.2.6 Event Sequence Modification

Modifications are currently saved as two sets of events: the set that was removed, and the set that was inserted. The only time that the events themselves are modified is during interval merging, where interval end points are changed to point to different events. At a higher level, modifications are currently maintained by the event categories they affect. Each event category maintains an ordered list of the modifications that have affected events of that category. A modification can be undone if it is the last modification that was performed for every event category that it involves.

This undo strategy is perhaps more nuanced than necessary. For the sake of consistency, it would probably make more sense to maintain modifications at the dataset level. That is, modifications can only be undone in the opposite order in which they were performed. This is the approach taken by most commercial applications in word processing and photo editing, and would likely be sufficient for this application as well.

On the back-end, modifications would be stored in a dedicated table that maintained the two lists of events. In this scheme, all modifications would have to consist only of an insertion list and a deletion list. Interval merging would have to be updated to match this format. This would mean that the Modification Table

could be structured as follows:

Modification ID	Insertion/Deletion	Event ID

This table would be maintained throughout a user's session with a given dataset. As the user performs various modifications, new events would be created in the Event Table, and the Record Event Table would be updated to remove deleted events and insert new ones. When users end a session, they would have the option to save the modified dataset, in which case deletion entries from the Modification Table would be permanently removed from the Event Table. The user could also choose not to save their modified dataset, in which case the inserted events would be removed from the Event Table and the Record Event Table, and the deleted events would be added back to the Record Event Table. Finally, the user could choose to save their modifications as a new dataset, in which case a new entry would need to be created in the Dataset table, the current dataset would need to be duplicated to that new dataset ID, and the modification list would need to be backed out of the former dataset. In all cases, the Modification Table would be cleared at the end of a session.

EventFlow would maintain the list of Modification IDs internally. The application would thus be responsible for the undo order as well as saving the modification process itself (in cases where users would like to repeat a series of modifications on a different dataset). The modification process could be stored on an external data source, but would likely not adhere to the standard, relational structure. It is also worth pointing out that this strategy for performing modifications is designed such that only one user can be accessing a dataset at any given time. This strategy would need to be revisited in situations where multiple users could be modifying a single dataset at a given time.

8.2.3 Display Operations vs. Data Operations

EventFlow currently supports two ways of changing the individual and aggregated visualizations:

- **Display Operations:** Display operations change the the way that EventFlow's underlying event structures are processed into the visualization, but do not change the event structures themselves. EventFlow's filter-based simplification controls, for example, qualify as display operations. When users filter a given event category out of the visualization, these events are simply skipped over in the preprocessing stage of the rendering. In no way are the events actually removed from their encapsulating records. Similarly, users can select any group of records to be removed from the display. Removed records are skipped over by the preprocessing code, but still remain a part of the overall dataset. Alignment qualifies as a display operation as well. Alignment enhances each event with a new timestamp, representing its location relative to the alignment point, but each event still maintains, and can easily revert back to, its original timestamp.

- **Data Operations:** Unlike display operations, data operations change EventFlow's underlying event structures. For example, if users execute a Find & Replace operation, the found events are removed from their encapsulating records, and the replacement events are inserted in their place. These new, altered records then get passed to the preprocessing code for rendering. Category-attribute swapping works in a similar way. Events of the selected category are removed, and replaced by new events that have the formers' attribute value as their category name.

EventFlow distinguishes very clearly between display operations and data operations in the underlying code base. However, the user-facing manifestation of this distinction is less clear. Currently, users can save their dataset, which will

save all of their data operations as well as their filter-based display operations. Users can also save a config file, which will save an additional subset of the display operations, including the color in which each event category should be rendered. Lastly, users can save a modification file, which will save the list of all the data operations, but not the display operations. There is currently no way to save a data alignment.

Additional work is needed to determine the best way to save both display operations and data operations in a way that best supports user needs. For example, if a user removes a set of records from the display based on a query that they perform, they may want to store that operation to be performed on subsequent datasets. However, removing that set of records is technically a display operation, and would not appear in the modification file. Similarly, users can choose to include events of a filtered category in a saved dataset, but can save a config file to indicate that the category should not be displayed in subsequent datasets.

There are also considerations on the data transformation front. Currently, users can perform Find & Replace operations on event categories that have been filtered out of the display. Should they be able to do this? From an accuracy perspective it may be better to only allow users to perform data operations on events that are being displayed. Overall, the application needs to provide users with more intuitive and more consistent mechanisms for distinguishing between these two types of operations.

8.2.4 Step-Up Complexity

Given the visual complexity metrics discussed in Chapter 6, EventFlow could be extended to automatically detect when the aggregated display is going to be too complex for users to discern meaningful event patterns. When this occurs, EventFlow could generate a simplified display, either by removing certain event categories or imposing a temporal range constraint on the display. Users could then incrementally add the complexity back in, making decisions about which events

are necessary as they go. This process could potentially feel more accessible to users than presenting them with an initial, complex display. Throughout my case studies, this initial display could be very overwhelming for users, and overcoming that response to take the first simplification steps was a tangible barrier that required some coaxing. It would perhaps be more effective to force users to start with a manageable amount of complexity, and step up from there.

8.2.5 High Quality Screenshots

In three years of work, there was only one feature that was requested by nearly every user, but was not implemented, and that was for EventFlow to be able to generate large, high quality screenshots of the aggregated display. Currently the best way to save the aggregated display is to take a screenshot of the desktop on which it is displayed. For many reasons, this is unideal, most notably when EventFlow is being run on a small display. There is nothing to prevent the software from being able to generate these screenshots internally. This would allow large screenshots to be generated, even when EventFlow is being run on a small display. It would also allow users to produce higher quality images for presentations and publications.

8.2.6 Sorting by Similarity

The integer programming query processing described in Chapter 5 would allow query results to be sorted according to how similar they are to the query sequence. This similarity could be determined by a variety of factors, and would ultimately be dictated by the objective function of the integer program. In this work, the integer programming back-end was prototyped within the EventFlow code base, but was never made visible to end users in the distributed executable. The next step of this implementation would be the design of an interface that would allow users to specify different query objectives, as well as testing to measure how beneficial

these ranking are in practice.

8.2.7 Interval Alignment

Previous work on the LifeFlow project already identified multiple alignment points as a potential avenue of future work [122]. EventFlow's inclusion of interval-based events only makes this point more salient. Currently, users can align a dataset around either an interval's start point or an interval's end point. However, this does not give users the same perspective of what happened both before and after an event as is does for point-based events. One end of the interval is always obscured in the aggregation. This difficulty is best illustrated by the LABA case study, where users had to look at the "step-up" and "step-down" prescription pattern separately. An interval alignment would allow users to simultaneously align a dataset by both the start and end point of a given interval. This capability would offer a much more clear perspective of what is happening before, during, and after the interval.

8.2.8 Time Granularity and Unknown Values

While this work has addressed the critical transformations that are needed to align event data to various objectives, it has not addressed the complexities that arise when data is either missing or contains events with drastically different time granularities. For example, EventFlow currently handles intervals with the same start and end time by incrementing the end time by one millisecond when the data is imported. This allows the underlying data structure to remain consistent, but does not address the possible effect that this can have on queries. Consider the following two events on input:

Record	Event Type	Start Time	End Time
342	Therapy	11/3/11	11/3/11
342	Sprained Ankle	11/3/11	

These two events would be translated into EventFlow as:

Record	Event Type	Start Time	End Time
342	Therapy	11/3/11 00:00:00:000	11/3/11 00:00:00:001
342	Sprained Ankle	11/3/11 00:00:00:000	

If users would then like to know if a patient sprained his ankle while he was at therapy, the answer is not clear in either formatting. The original formatting only indicates that these two events happened on the same day, and EventFlow's formatting suggests that the patient sprained his ankle at exactly the start time of his therapy. Neither of these outcomes are ideal.

These time granularity issues, as they relate to querying, remained outside the scope of this work due to the fact that they were not a pressing concern of my case study partners. However, data inaccuracies will arise in real datasets, particularly as more and more researchers analyze pre-existing data, rather than data that they directly collected. Two questions stem from these errors:

- How do you visualize uncertainty or potential inaccuracies?
- How can these errors be addressed through data transformation?

The first of these questions would likely lead to an entirely new avenue of future work. The second of these questions could likely be addressed by expanding the metric constraint capabilities of EventFlow's Find & Replace system. Users could specify replacements with more detailed relationships to the original events.

This capability could be used to address both time granularity and unknown value errors.

8.3 Summary

This dissertation presents a series of strategies for leveraging visualization across a wider range of the data analysis spectrum. Though this work is specific to temporal event data, it lays the groundwork for how similar approaches might be taken with different data types. The contributions of my work have been implemented in a readily accessible tool, and tested in collaboration with a wide variety of real work researchers and practitioners. In this chapter, I also provide suggestions for how this tool might be extended to meet both commercial and research needs. Ultimately, this work is intended to highlight the practical challenges of real-world datasets and the steps that must be taken to overcome them in the pursuit of novel insights.

Appendix A

The Aggregated Display Construction

In this Appendix, I provide a step-by-step tutorial of how the EventFlow aggregated display is constructed, according to the design of Wongsuphasawat [122]. I use a basketball play-by-play dataset as an example, which is designed to both better illustrate the display construction, as well as demonstrate how very long event sequences can be broken down into usable increments.

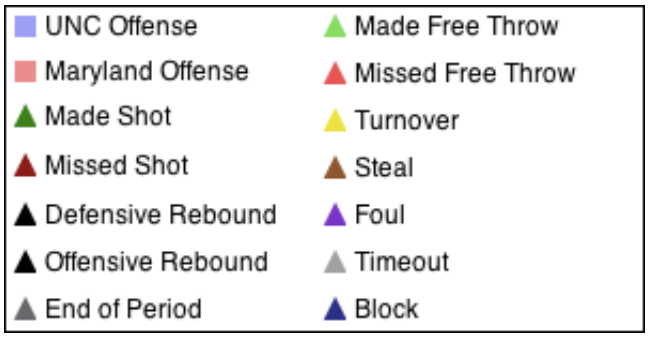


Figure A.1: Legend of Play-By-Play event categories.

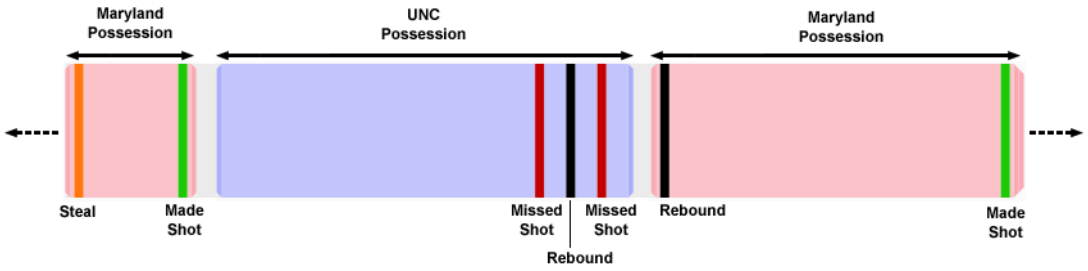


Figure A.2: **Step 1 -** A basketball game can be thought of as a series of alternating possessions. Each possession has a duration, and contains the sequence of play-by-play events that occurred during that possession.

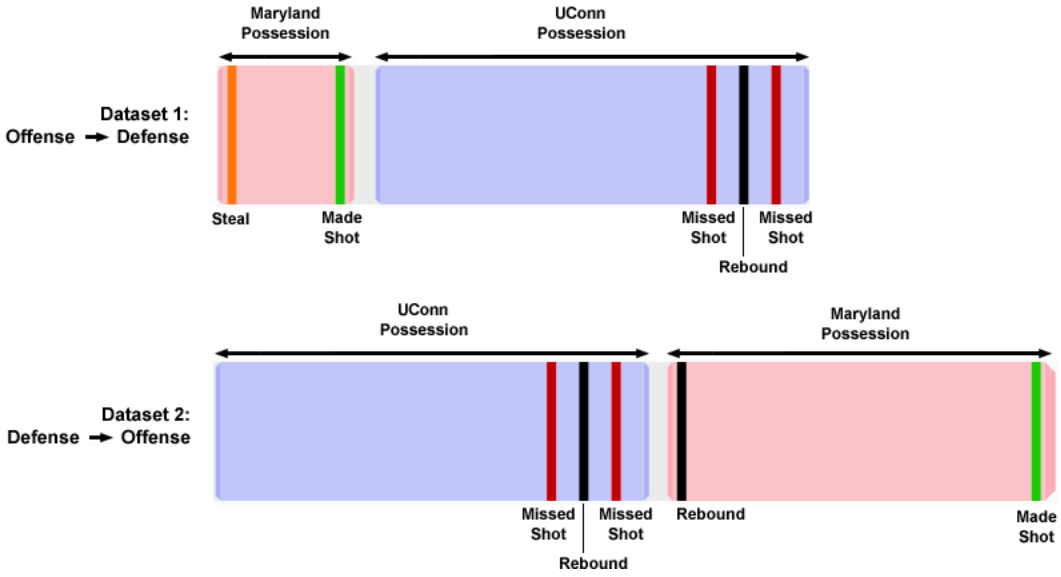


Figure A.3: **Step 2 -** The game is broken down into two overlapping datasets. The first dataset breaks the game into individual records consisting of an offensive possession followed by a defensive possession. The second breaks the game into individual records consisting of a defensive possession followed by an offensive possession.

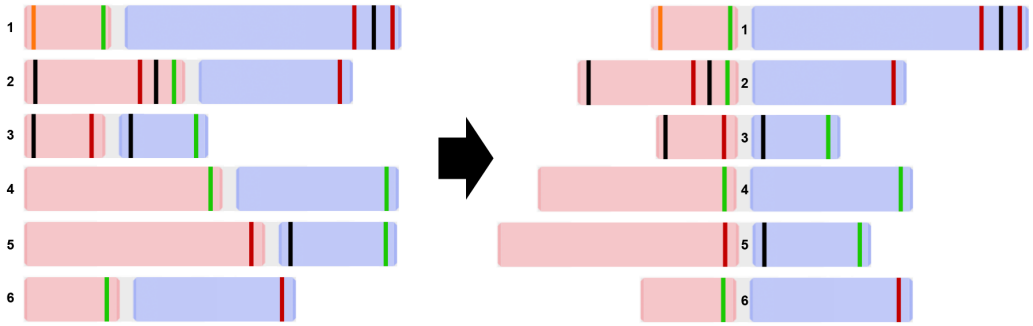


Figure A.4: **Step 3 -** A dataset of two-possession increments (in this case the Offense → Defense dataset) is aligned by the point of possession change. The default alignment is the start of each record, but this is the alignment that best suits these basketball datasets.

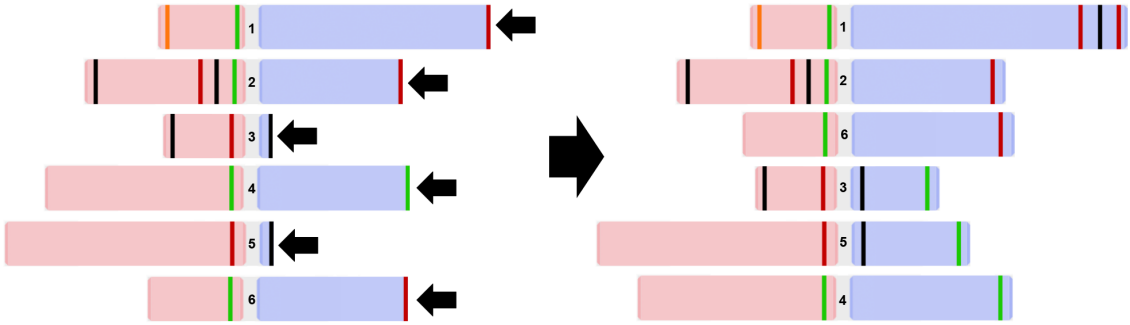


Figure A.5: **Step 4 -** We look at the first event to the right of the alignment. The records are sorted according to the most common event. In this case, the most common first event is a missed shot. (The sorting can also be done by the first event to the left of the alignment.)

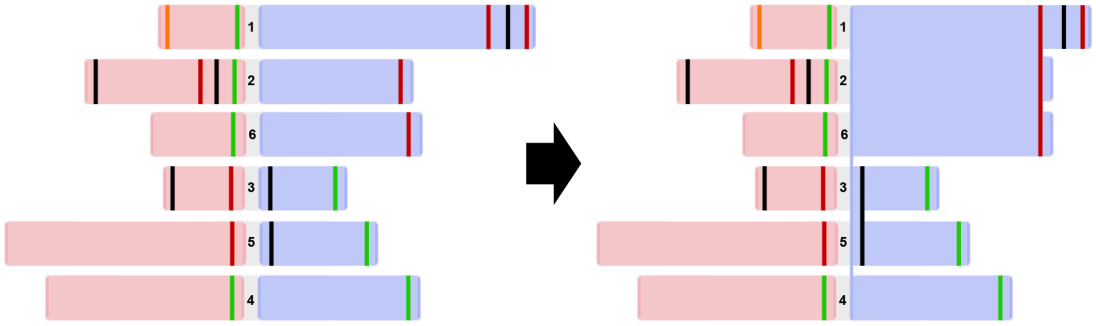


Figure A.6: **Step 5 -** Sorted events are consolidated into groups, the grouped bar is placed horizontally (on the time axis) by averaging the horizontal placement of its composite events.

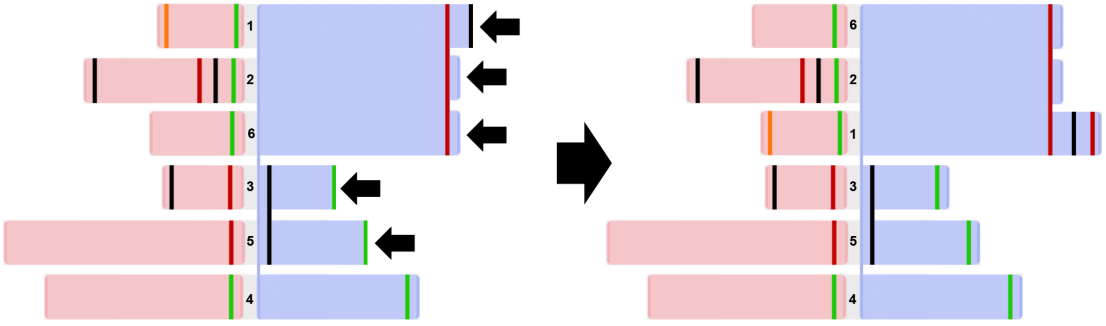


Figure A.7: **Step 6 -** For groups consisting of more than one record, we continue on to the next event, and again, sort the records within each group according to the most frequent next event. In this case, only the top-most group requires this additional sorting.

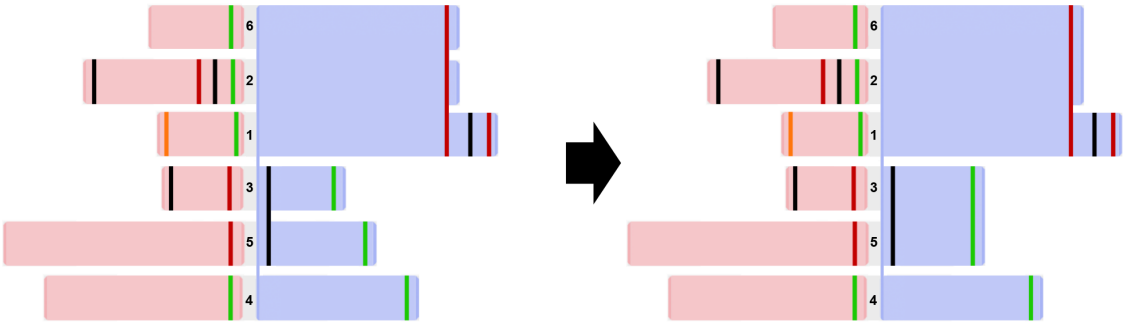


Figure A.8: **Step 7 -** Again, sorted events are consolidated into grouped bars.

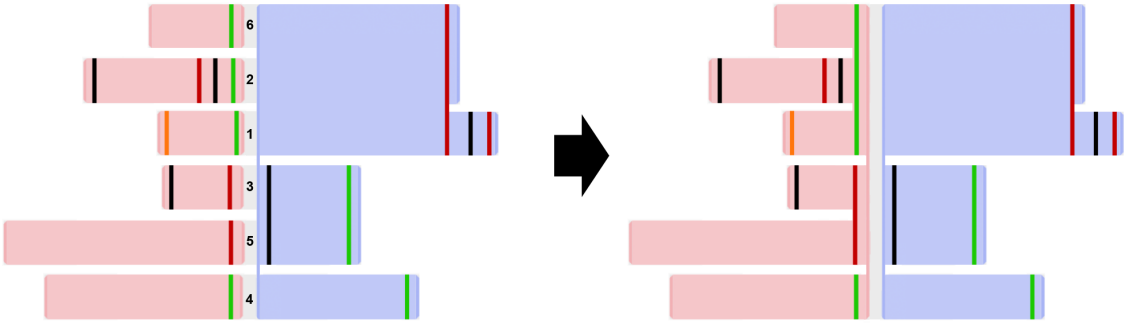


Figure A.9: **Step 8 -** The aggregation is finished by consolidating the events to the left of the alignment as best as possible.

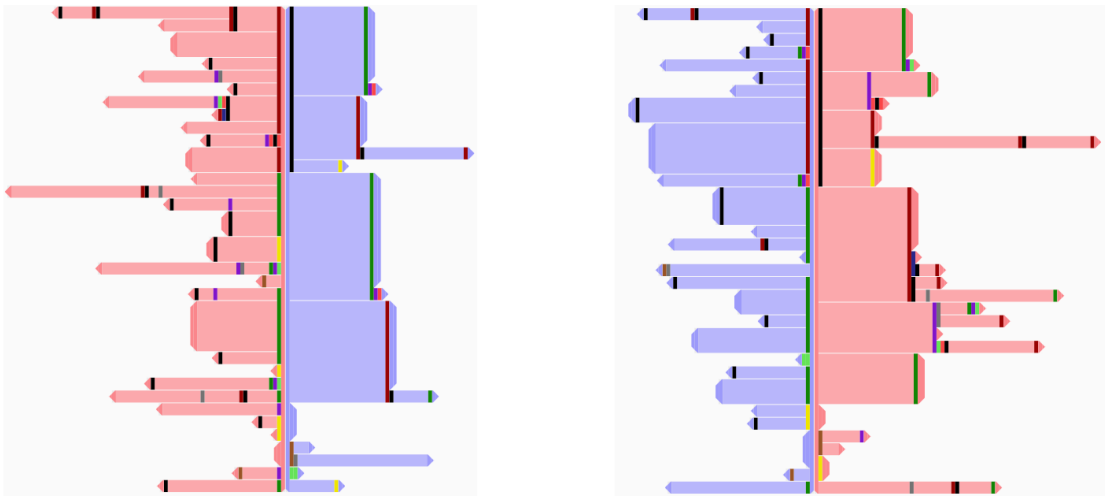


Figure A.10: **Final -** The Offense → Defense dataset (left) and the Defense → Offense dataset (right). The height of each bar corresponds to the number (actually, the percentage) of records that comprise it, and the horizontal axis is time.

Appendix B

Query Graphic Design Study

The goal of this design study was to understand how users expect to define a query with a visual query language for categorical temporal event data. The study was conducted between September 3th and September 14th, 2012, and supervised by Juan Morales del Olmo. The 20 participants were computer science graduate students who self-rated as expert computer users, but with limited experience with databases, command-based query formulation, and temporal event data. The participants were asked to draw on paper with colored pencils a list of 7 queries. They were then asked to run the same list of queries in an evolving mock-up of EventFlow's advanced query system. These tasks were completed by almost all the participants in less than 45 minutes.

B.1 The Script

All participants were read the following script at the beginning of the study. The script was designed to provide context for the queries that the participants were being asked to design:

Imagine you are a doctor and need to search for patient records with some particular patterns.

(Participant is shown the legend and a example records shown in Figure B.1.)

The goal of this study is to create a query language for searching event records that

uses visual elements similar to those that are used to display the records. There are two types of events: point events (Stroke, Admitted and Diagnosed) and interval events (Drug A and Drug B). Interval event categories are represented as a diamond in the legend, and a double-sided arrow in the event records. Point events are represented as triangles in both settings.

(Participant is given a blank piece of paper.)

Here you have a piece of paper where you will draw the queries choosing the visual elements you want. For example, if you are interested in patients that had a Stroke, then were Admitted, and then were Diagnosed, you would draw a green triangle, then a blue triangle, then an orange triangle.

(Participant is shown this simple, introductory example.)

The study consist of 3 steps. In the first step, you will design the queries by drawing on paper. Then, in the second step, you will define the same 7 queries using a computer interface. Finally, you will have time to suggest improvements and other ideas, or explain your decisions. Please feel free to speak aloud while drawing, as it is helpful for us to know your thoughts.

Remember that you don't have to draw the results of the query, you are asked only to draw the query using those visual elements and others that you could imagine, like annotations, text or whatever.

Do you have any questions or requests before we begin?

B.2 Queries

Query 1: The patient is Admitted, then begins taking Drug A. The patient has a Stroke while taking Drug A.

Query 2: The patient takes Drug A for at least 3 days.

Query 3: The patient has a Stroke, then starts taking Drugs A within 5-10 days after having the Stroke. To clarify, there is a gap between those events.

Query 4: The patient has a Stroke, then is Admitted.

Query 5: The patient has a Stroke, then is Admitted. The patient is NOT Diagnosed in between the Stroke and being Admitted.

Query 6: The patient is Admitted, then has a Stroke. The patient is NOT taking Drug A when the stroke occurs.

Query 7: The patient is not taking Drug A at some point before having a Stroke.

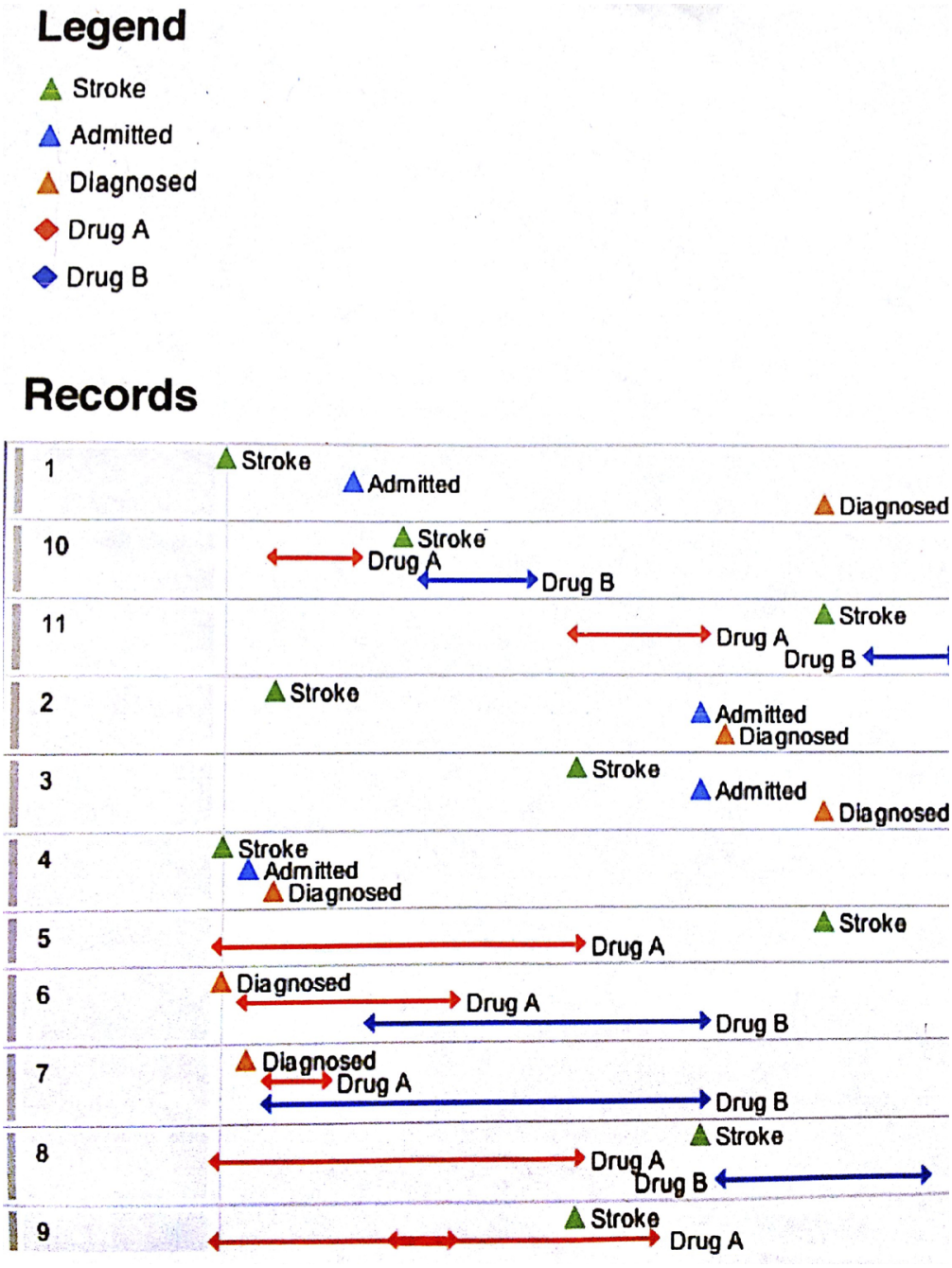


Figure B.1: Participants were shown the EventFlow legend and a set of simple sample records.

B.3 User Responses

Listed below are the verbal clarifications, questions, and descriptions provided by each of the participants, followed by the drawings they produced:

Subject 1

- **Query 1:** The subject re-draws the query to express that the query does not specify an end, only the beginning of Drug A.
- **Query 3:** Here again the end of the interval is not specified, but this time the subject does not draw anything at the end of the line.
- **Query 6:** The Stroke is surrounded to express a relation. The crossed diamond refers to the Stroke to express "not - when - occurs."

Subject 2

- **Query 2:** The non-filled arrow means "at least." No duration specified.
- **Query 3:** The non-filled first arrow means "not certain at this point," and the final filled arrow means "not beyond."
- **Query 5:** The subject tries to express the absence of "Diagnosed" drawing the optional presence of "Diagnosed" after or before.
- **Query 6:** The user finds it difficult to define this query and be consistent with the previous symbols, so introduces new ways of express the absence and the optional presences. Introduces a double triangle symbol to express absence. In a second the subject sketches a non-filled circle for the same propose.
- **Query 7:** The subject is chaotic in this query definition. The subject starts reusing the double triangle symbol but changes it to a circle because wants

to express that there is a gap in the administration of Drug A or maybe the patient never took the drug before having the "Stroke." The circle represents the gap in the administration.

Subject 3

- **Query 1:** The subject does not want to specify the end of the interval.
- **Query 2:** Again, the subject does not want to specify the end of the interval.
- **Query 3:** The uncertainty is expressed with multiples arrows or with one arrow inside square brackets.
- **Query 4:** The time is optional if there is no ticks in the time line.
- **Query 6:** A crossed beginning of a red interval expresses that "the patient is NOT taking Drug A when the stroke occurs."
- **Query 7:** The dashed line expresses the discontinuity in the administration of the "Drug A." The question marks the uncertainty.

Subject 4

- **Query 1:** The subject does not want to specify the end of the interval.
- **Query 2:** Again, the subject does not want to specify the end of the interval.
- **Query 3:** The subject does not want to draw the interval line after the 10th day.
- **Query 5:** I have to clarify that the specification of the absence is needed. Finally the subject defines a new horizontal band for the absences, like in the current implementation. So the absence of a point is an interval.

Subject 5

- **Query 5:** The absence is represented with a cross below the event, and uses a new line for placing the absence indications.

Subject 6

He had read the Infovis paper, so was biased. All the drawing is similar to our design.

Subject 7

- **Query1:** The subject perceives the arrows as "the line continues in that direction." Also specifies that there is a gap between the two events.
- **Query 2:** The subject draws a double bar at the end of the line, represents a rubber band that acts as a "doorstop." This rubber prevents the line from being shrunk.
- **Query 3:** The same concept of the rubber band explained in more detail. The final answer is the one marked with the star. Actually, the line only had one rubber band at the beginning of the line. The second one was added to explain the effect of moving the line to the right.
- **Query 5:** The subject defines a new horizontal line for the absences, like in the current implementation. So the absence of a point is an interval.
- **Query 6:** The absence of "Drug A" is drawn explicitly as a diamond, otherwise (drawn like a line) would express that there is a gap before and after the "Stroke" while the patient is not taking the drug.
- **Query 7:** Very confused with this question. The subject feels that a new event is needed to express this query. So introduces the event "Pre-

scribed/Recommended Drug A," and after that there is an interval while the absence of "Drug A" occurs (again a diamond).

Subject 8

- **Query 1:** The subject does not want to specify the end of the interval.
- **Query 2:** Again, the subject does not want to specify the end of the interval. The dashed line expresses uncertainty.
- **Query 3:** The dashed line expresses uncertainty.
- **Query 5:** The absence is represented with the negation of an interval, represented with a striped band.
- **Query 7:** The final answer is the second one.

Subject 9

- **Query 2:** The dashed line expresses "at least."
- **Query 5:** I have to clarify that the specification of the absence is needed. The absence of a point event is a non-filled triangle.
- **Query 7:** The absence of "Drug A" at some point is specified as a gap in the line.

Subject 10

- **Query 3:** I have to remind that the specification of the 10 days at most is needed.
- **Query 5:** I have to clarify that the specification of the absence is needed. The absence of a point event is represented as a cross and a wriggled interval.
- **Query 6:** The absence of the interval is represented as a surrounded cross.

- **Query 7:** The valid green triangle is the last one.

Subject 11

- **Query 2:** "At least" is represented with a left bar instead of a left arrow.

Subject 12

- **Query 1:** The subject does not use the lineal representation of the interval, always uses the diamond.
- **Query 2:** The dashed line expresses uncertainty.
- **Query 3:** The expression of the gap is represented in the time line. The first dashed segment means "at least" and the second "at most."
- **Query 6:** The crossed thing in the middle is a mistake. The absence of "Drug A" is represented as a crossed diamond.
- **Query 7:** The absence of "Drug A" is represented as a crossed diamond. And the line means that the absence occurs anywhere in this space.

Subject 13

- **Query1:** Instead of using a line for the interval events, the participant uses a wider arrow in order to include inside the Stoke triangle.
- **Query 2:** The big cross represents at least.
- **Query 3:** The gap is represented as a black interval with time marks.
- **Query 5:** At this time the subject adds a little arrow between all the events to express the sequence. The absence is represented with a no entry signal.
- **Query 6:** Now the subject introduces plates under the events represents chunks of time. So two events in the same plate occurs in the same time.

- **Query 7:** The final answer is the second one.

Subject 14

- **Query 2:** The subject adds an extension of the interval. It is a dashed line to express uncertainty.
- **Query 3:** This time the participant does not want to close the interval.
- **Query 4:** The subject assumes that there is no gap between the events. This is represented with the ticks in the time line.

Subject 15

- **Query 3:** The gap is represented with a blank between the two events. The time is specified in the time line, and the red point that is in the final drawing was accidentally added while explaining the design choices. Adds to the time line the word "Days."
- **Query 7:** The subject wants to express that the patient could take the drug or not take it at all.

Subject 16

- **Query 5:** I have to clarify that the specification of the absence is needed. The correct answer is the second one.
- **Query 7:** The subject wants to express that the patient "maybe" takes the drug or maybe never takes it at all.

Subject 17

This subject apparently understood everything right away.

- **Query 1:** After my pointing out that he was not drawing the events "at the same time."
- **Query 2:** The subject specifies "3 days" with 3 different intervals.
- **Query 5:** The subject express the absence of "Diagnose" with a crossed "D."
- **Query 6:** In order to specify that "the patient is NOT taking Drug A when the stroke occurs," the subject draws the left arrow of the interval just under the triangle.
- **Query 7:** The participant includes a curved interval representation to express that this event is optional (maybe occurs or maybe not).

Subject 18

- **Query 1:** The red arrow means "then." The "Drug A" is represented with a triangle followed by an interval.
- **Query 2:** Here the "Drug A" is a triangle, and the duration is represented an interval with only the right arrow.
- **Query 6:** The subject draws a second answer to express that the absence of "Drug A" is only in the same moment that the "Stroke" occurs.

Subject 19

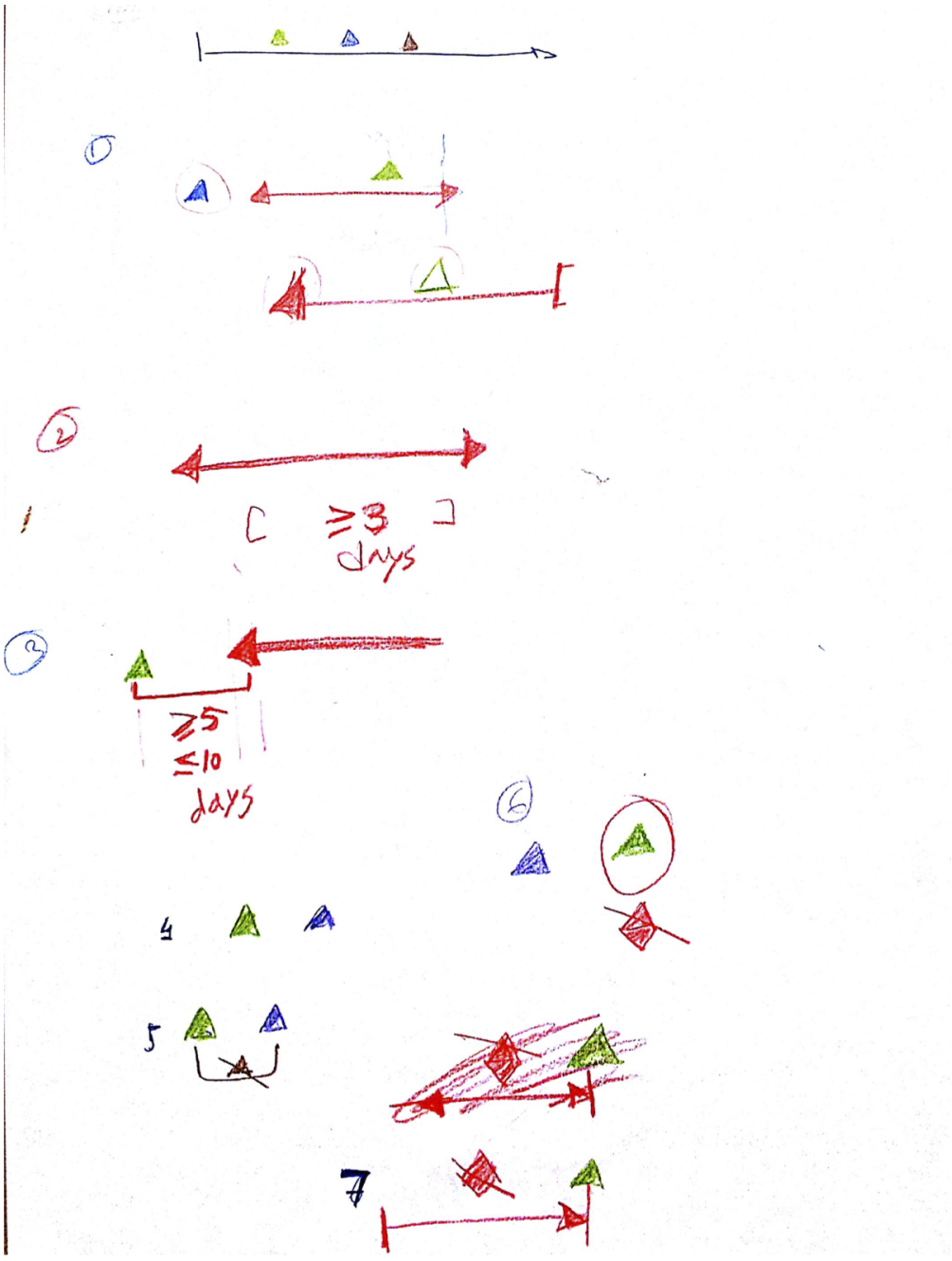
- **Query 3:** I need to remind that the time has to be specified.

Subject 20

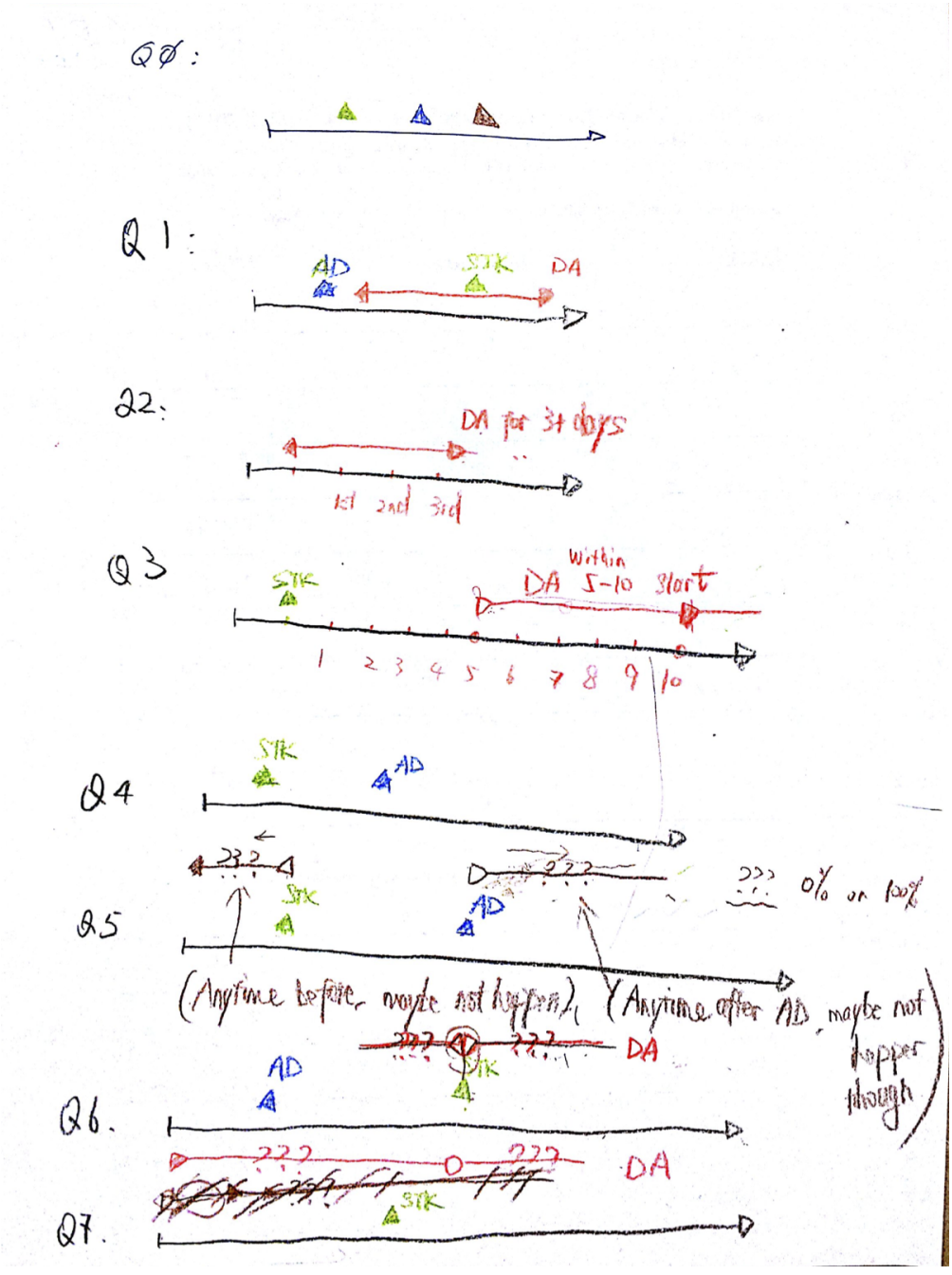
- **Query 2:** The subject draws one interval for one day, and does not specify the "at least 10 days."

- **Query 5:** The participant draws a diamond as the absence of a point event.
- **Query 6:** The participant draws a "*" as the absence of an interval event. Again, one tick in the time line represents one day.
- **Query 7:** For this subject, the time scale is very important, so to define this query only comes with a "query by example" design. In this example, the patient did not take "Drug A" in the second day.

Subject 1:



Subject 2:


Q0:
Q1:
AD
STK
DA
Q2:
DA for 3 days
1st 2nd 3rd
Q3
STK
Within
DA 5-10 short
1 2 3 4 5 6 7 8 9 10
Q4
STK
AD
Q5
STK
AD
??? 0% or 100%
(Anytime before, maybe not happen)
(Anytime after AD, maybe not happen though)
DA
Q6.
AD
DA
STK
Q7.

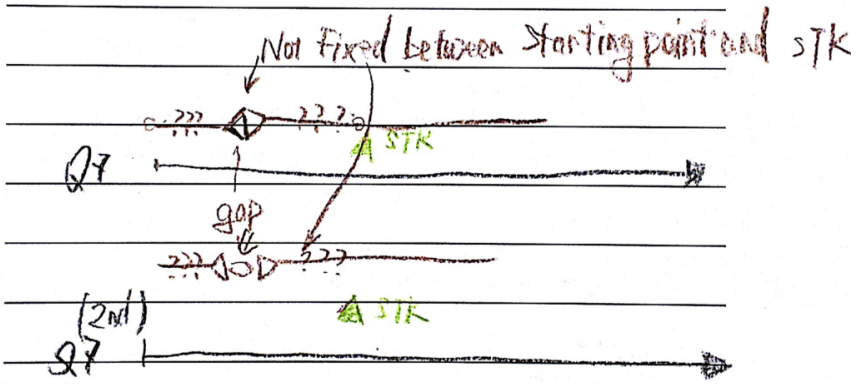
Subject 2 (cont.):

R1

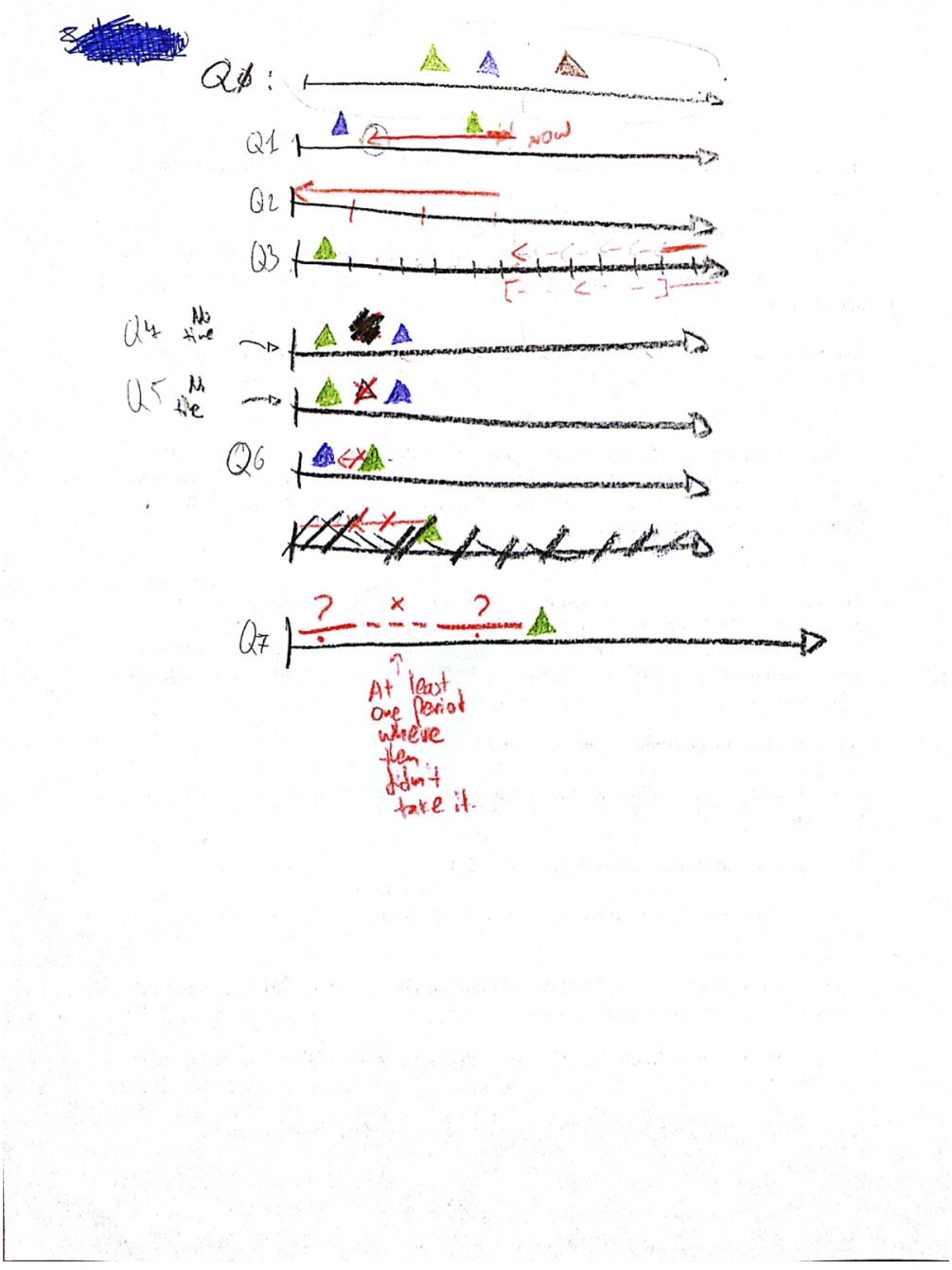
User Study: Visual Query Language User Interface (Fall 2012)
HCIL – Catherine Plaisant, Megan Monroe, Juan Morales
IRB Protocol: 351899-2 - User Interfaces for browsing and searching temporal information

I acknowledge receiving $20 for participating in this user study:

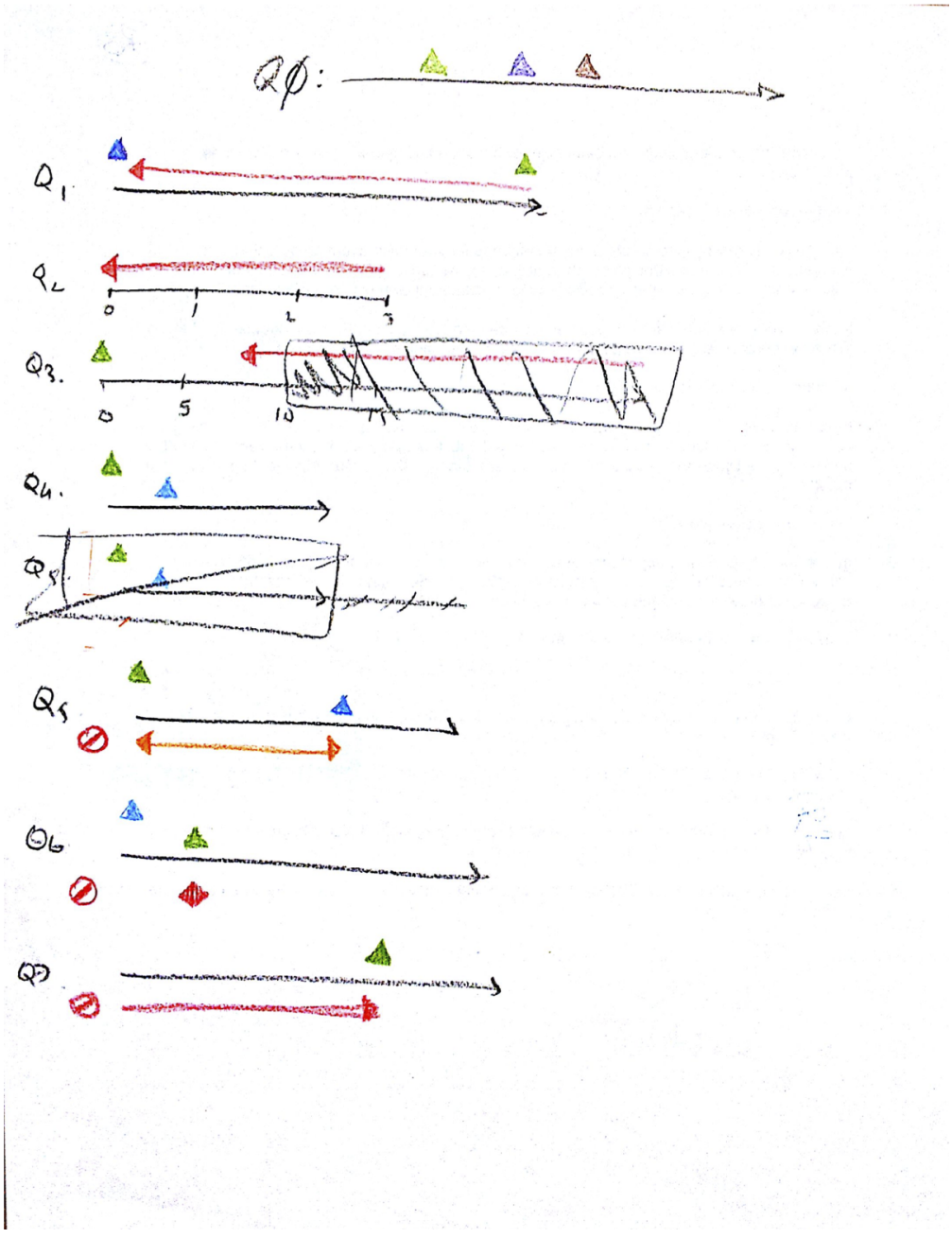
NAME **Signature** **Date**



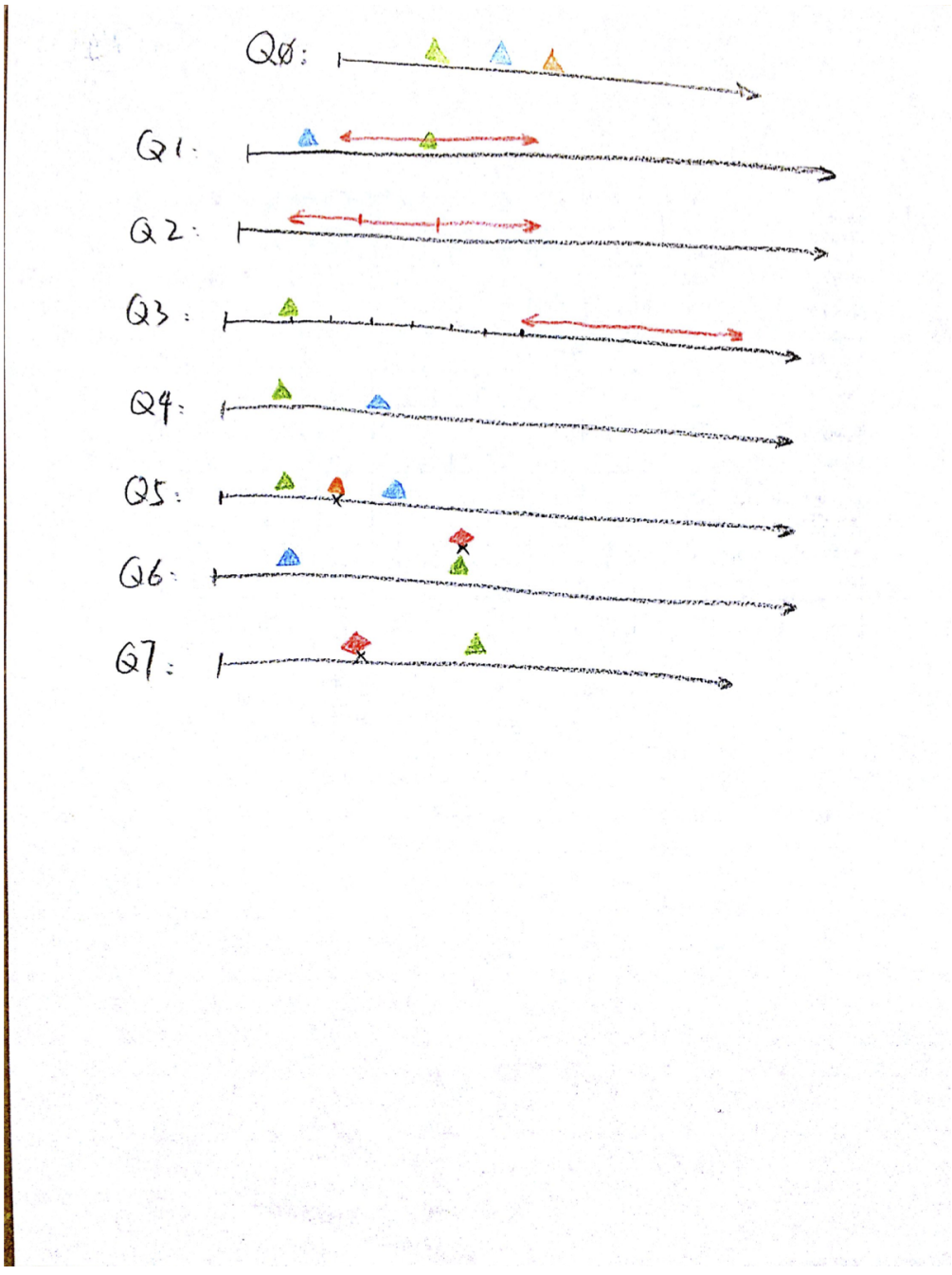
Subject 3:


Q0 :
Q1
now
Q2
Q3
Q4 No time
Q5 No time
Q6
Q7
?
x
?
At least one period where they didn't take it.

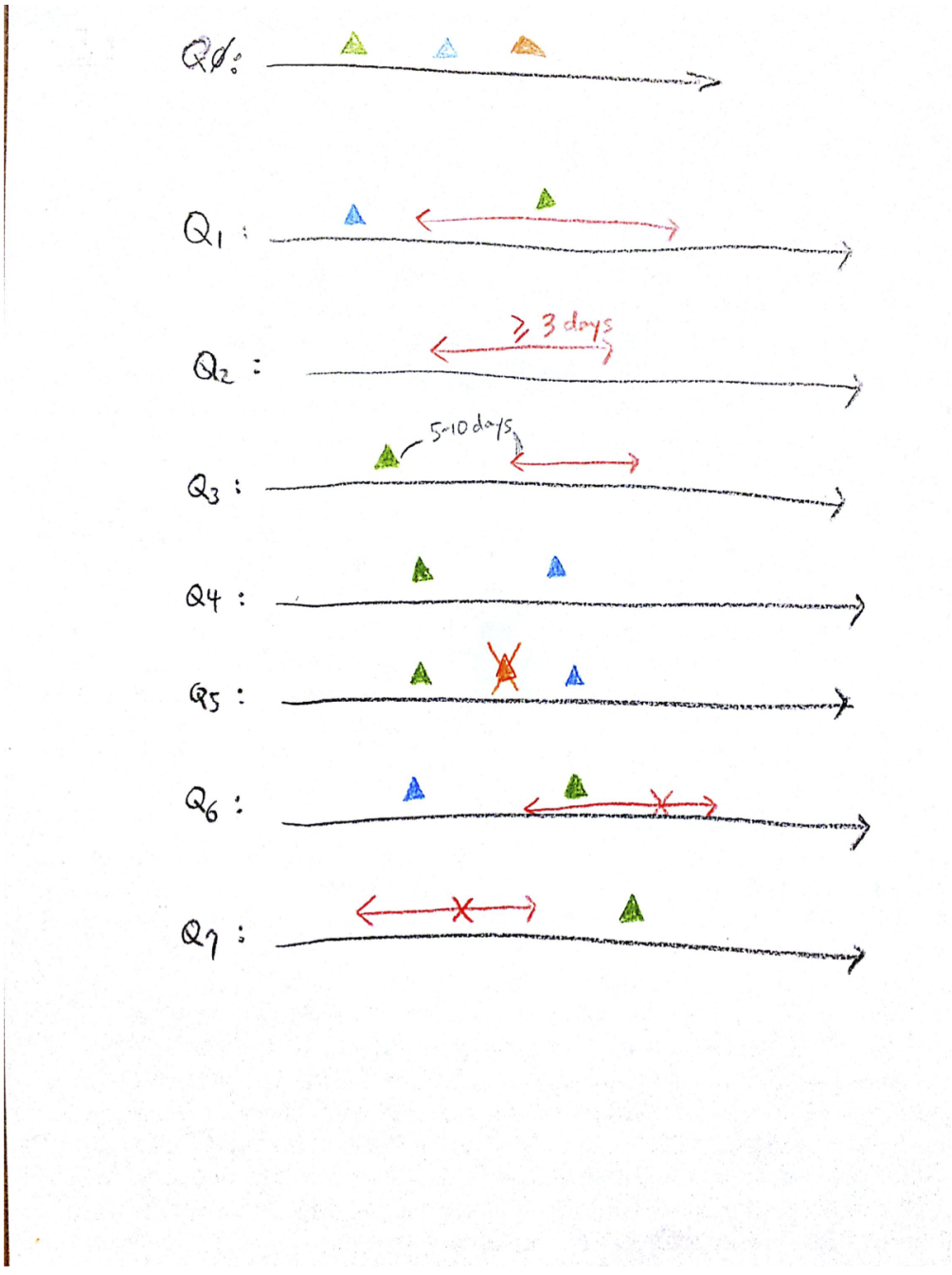
Subject 4:



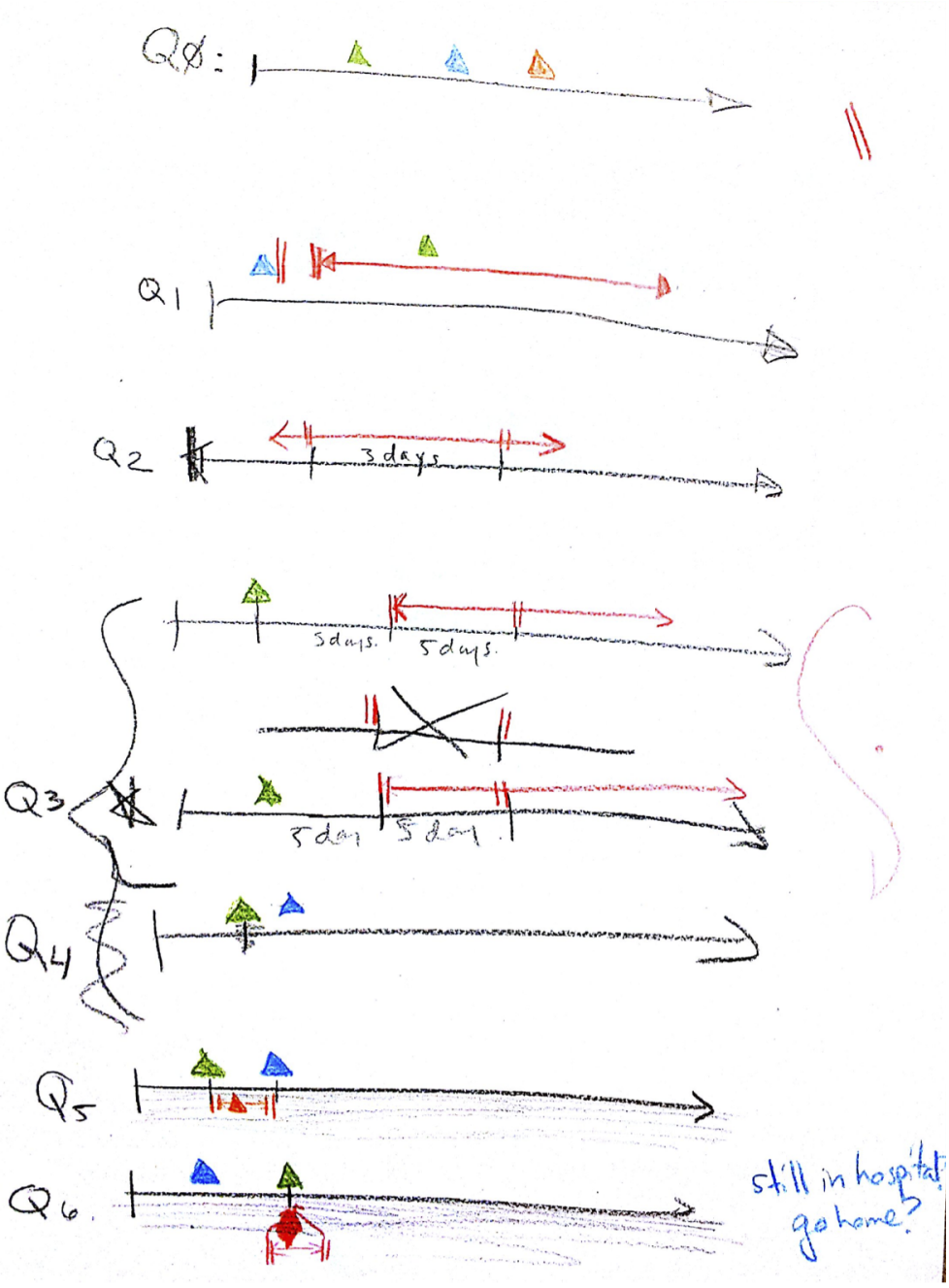
Subject 5:



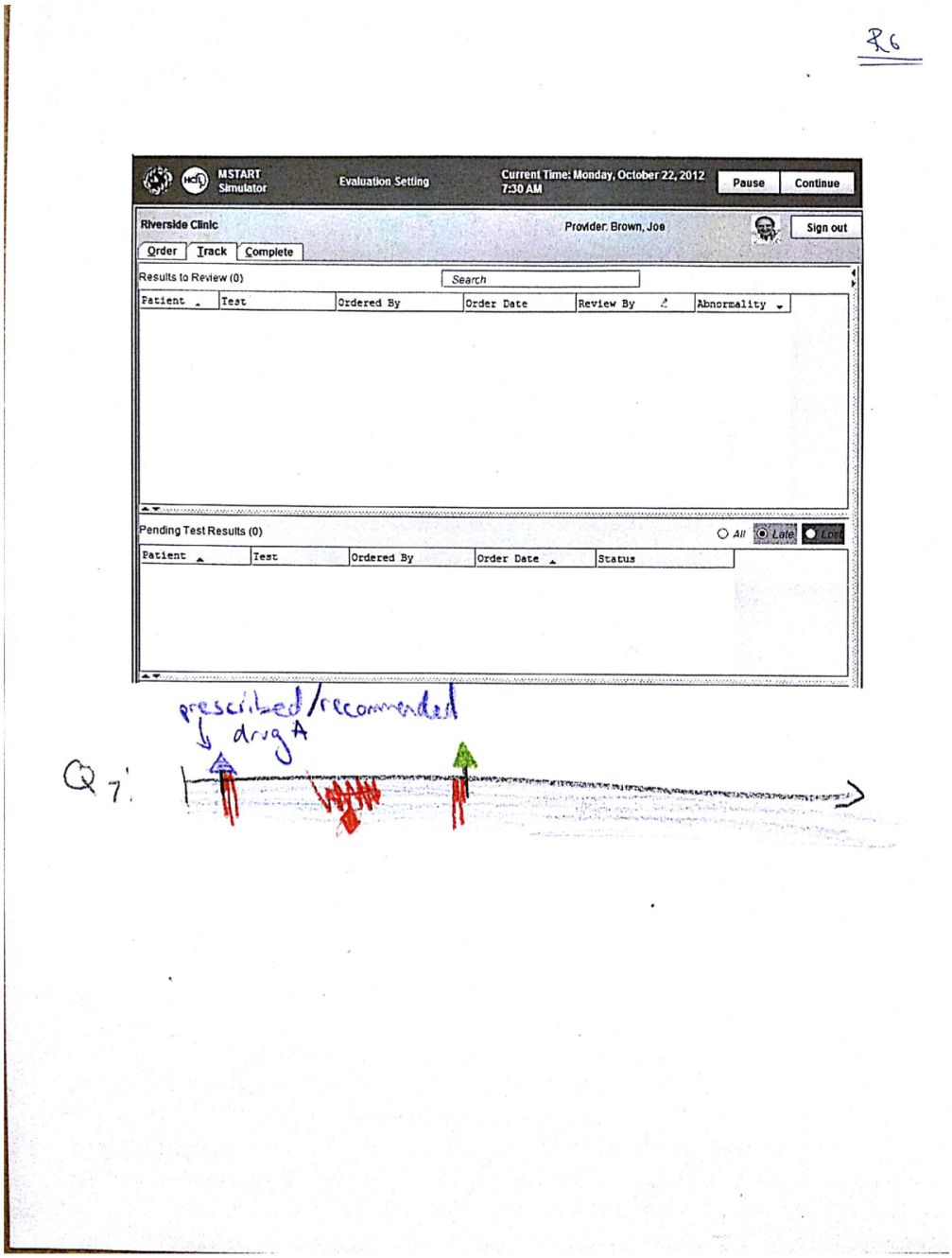
Subject 6:



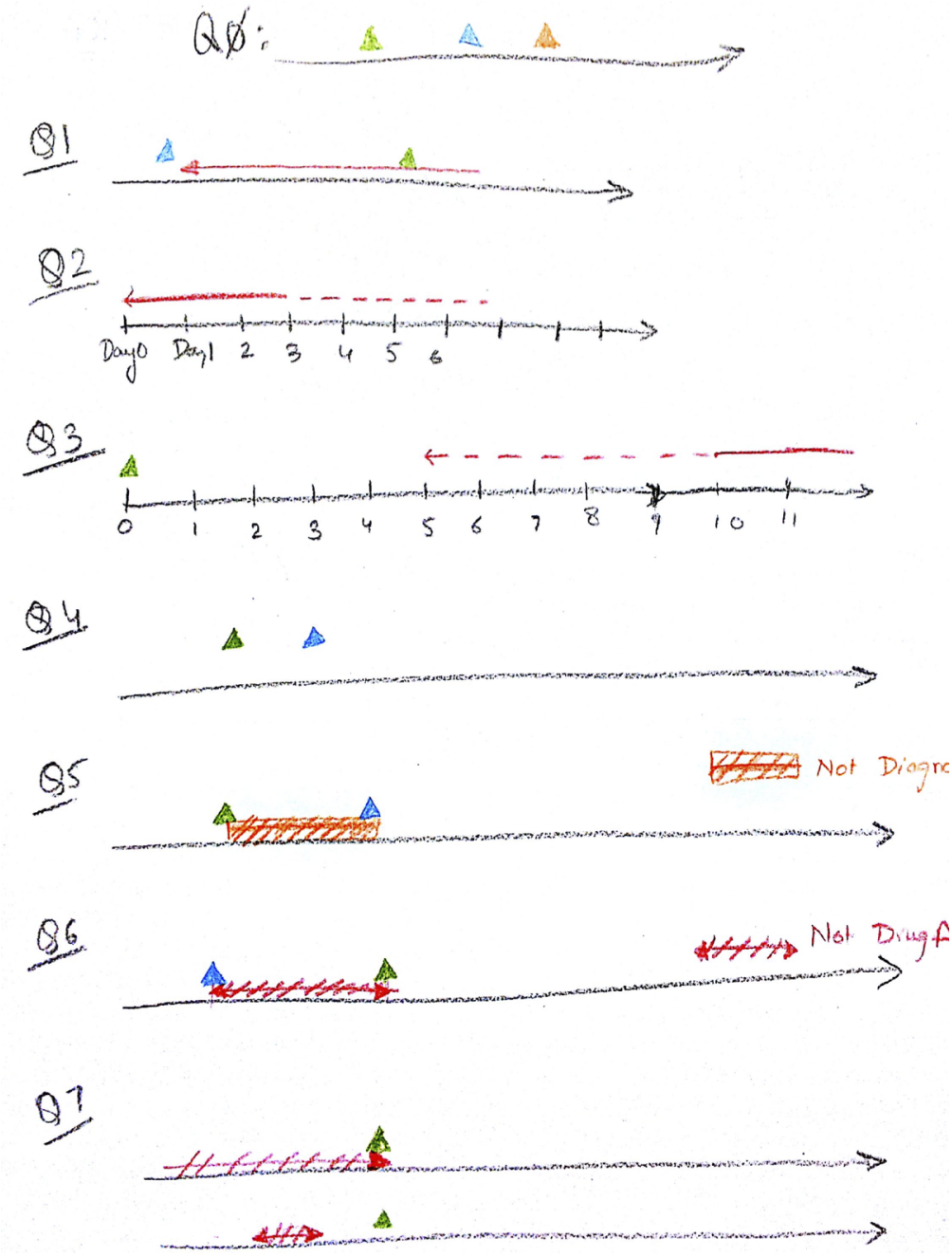
Subject 7:


Q0:
Q1
Q2
3 days
5 days.
5 days.
Q3
5 day
5 day.
Q4
Q5
Q6.
still in hospital
go home?

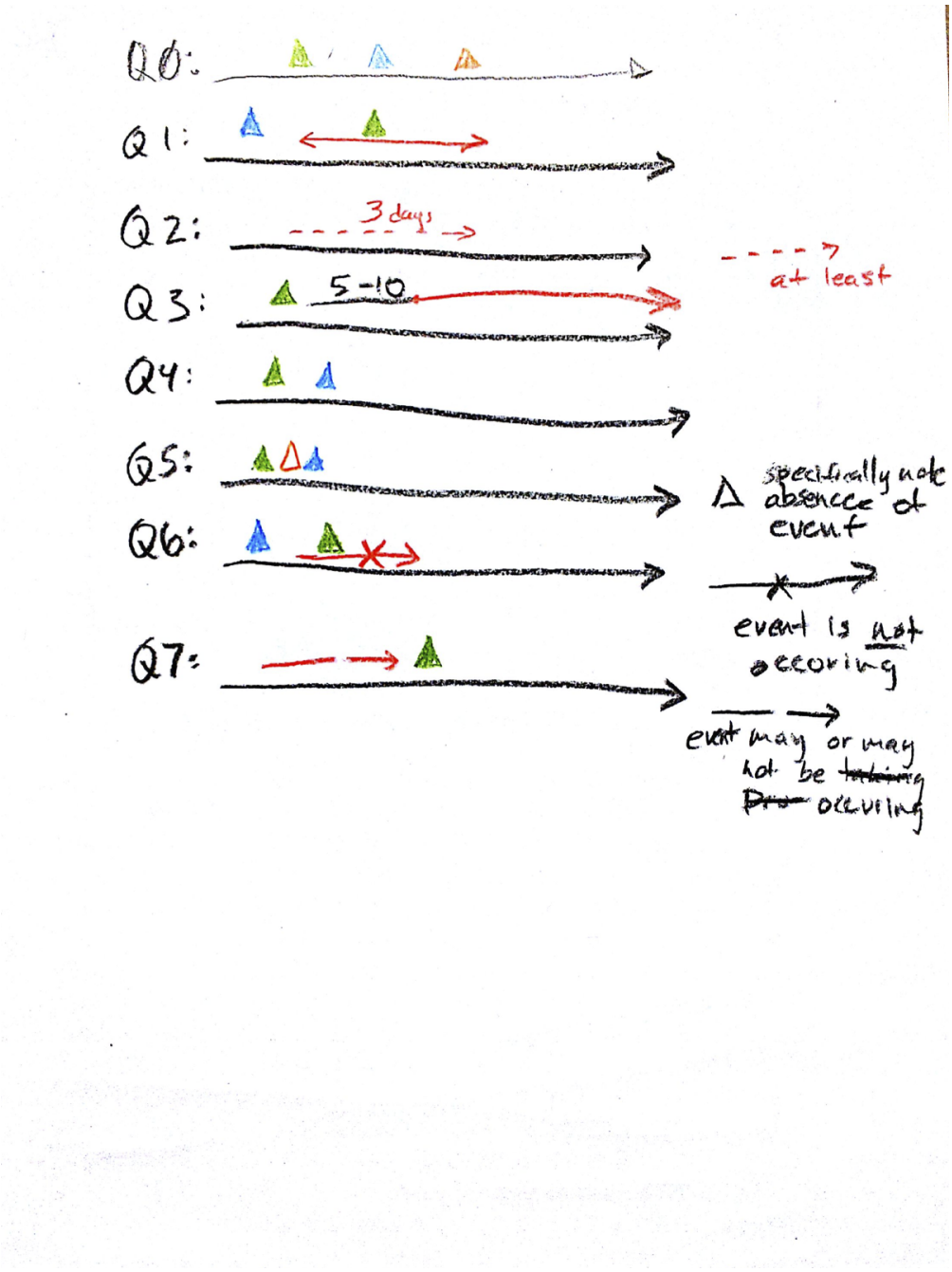
Subject 7 (cont.):



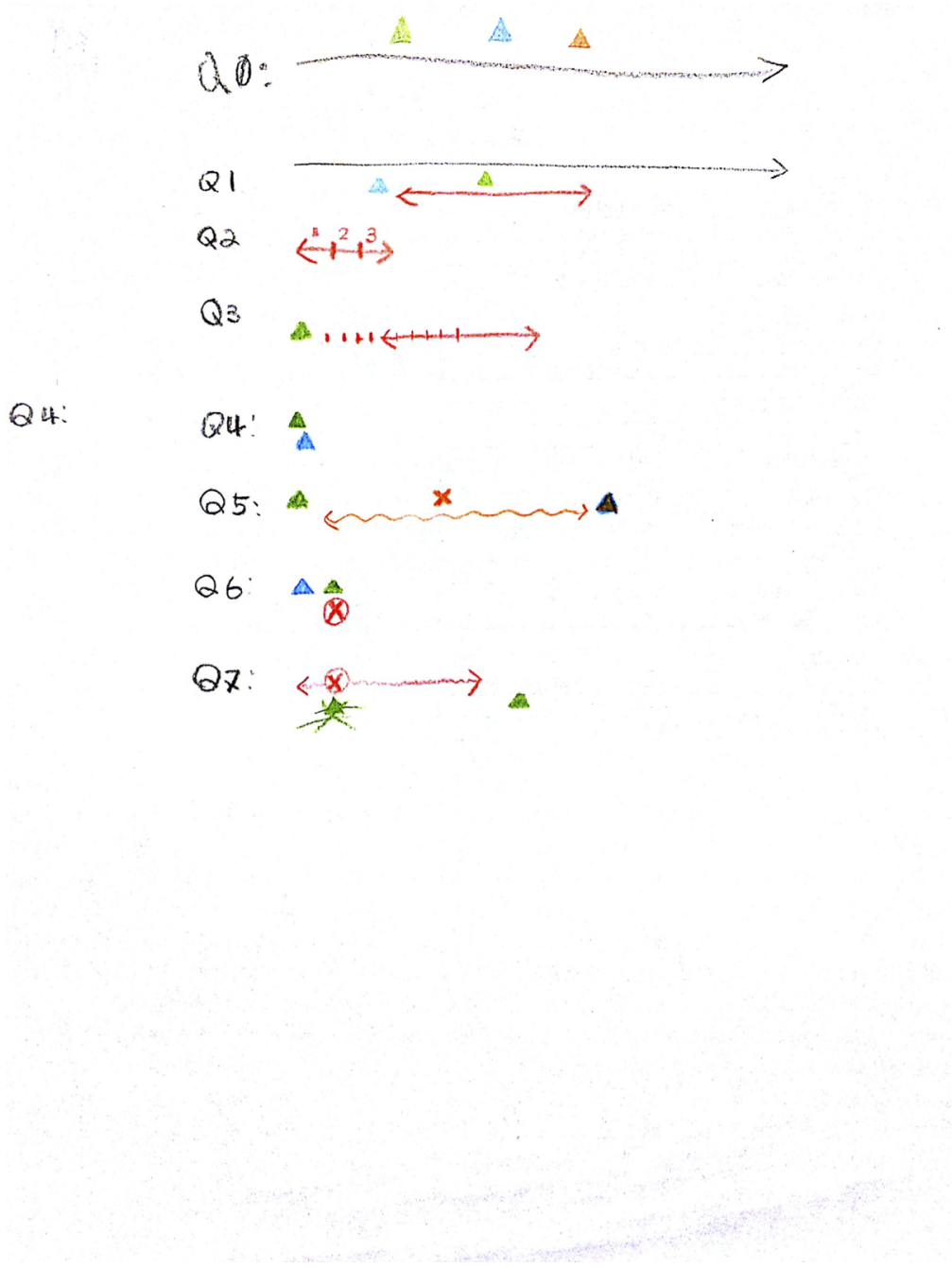
Subject 8:


Q0:
Q1
Q2
Day0 Day1 2 3 4 5 6
Q3
0 1 2 3 4 5 6 7 8 9 10 11
Q4
Q5
Not Diagno
Q6
Not Drug-f
Q7

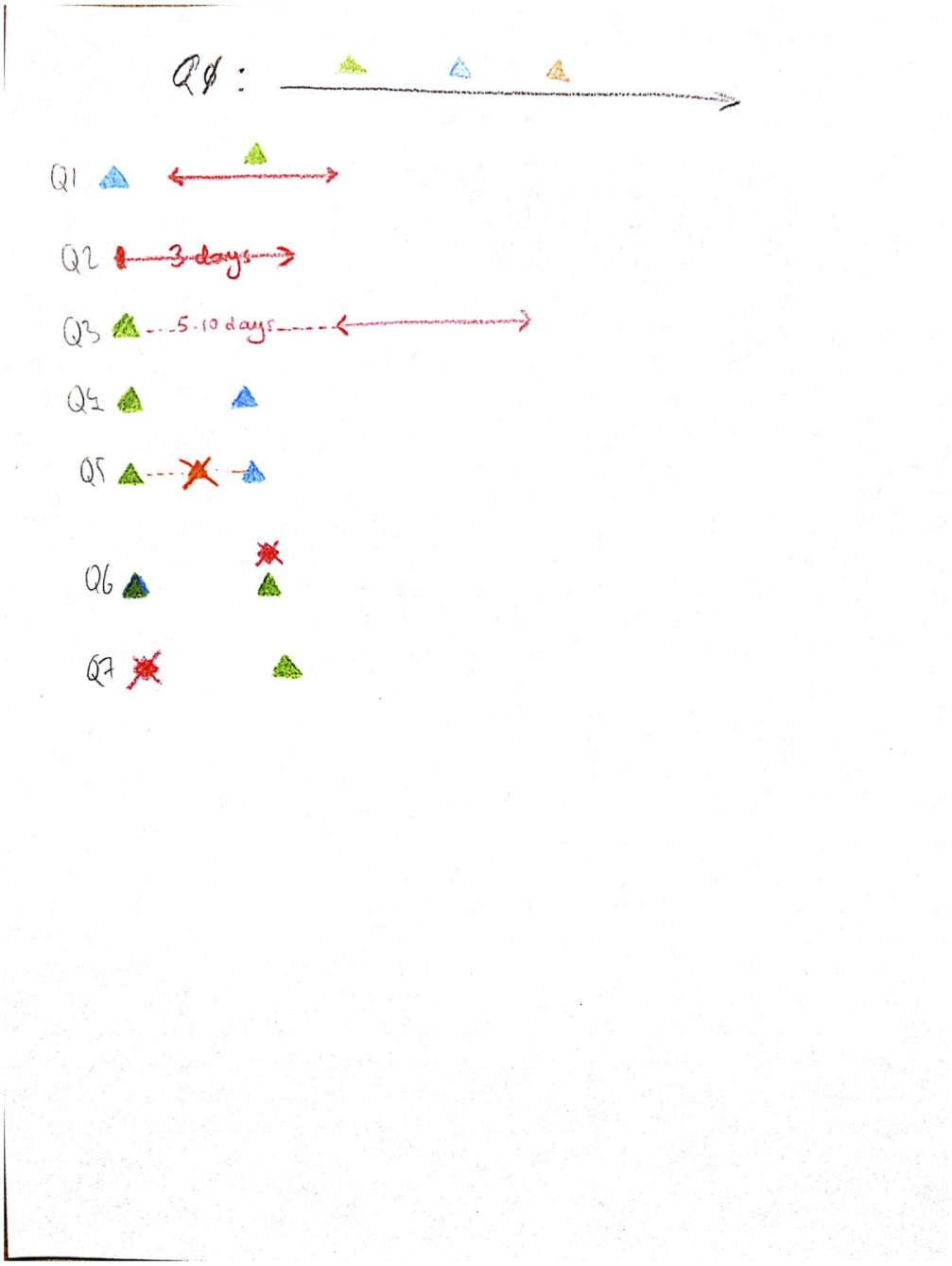
Subject 9:


Q0:
Q1:
Q2:
3 days
Q3:
5-10
at least
Q4:
Q5:
specifically note absence of event
Q6:
event is not occuring
Q7:
event may or may not be occuring

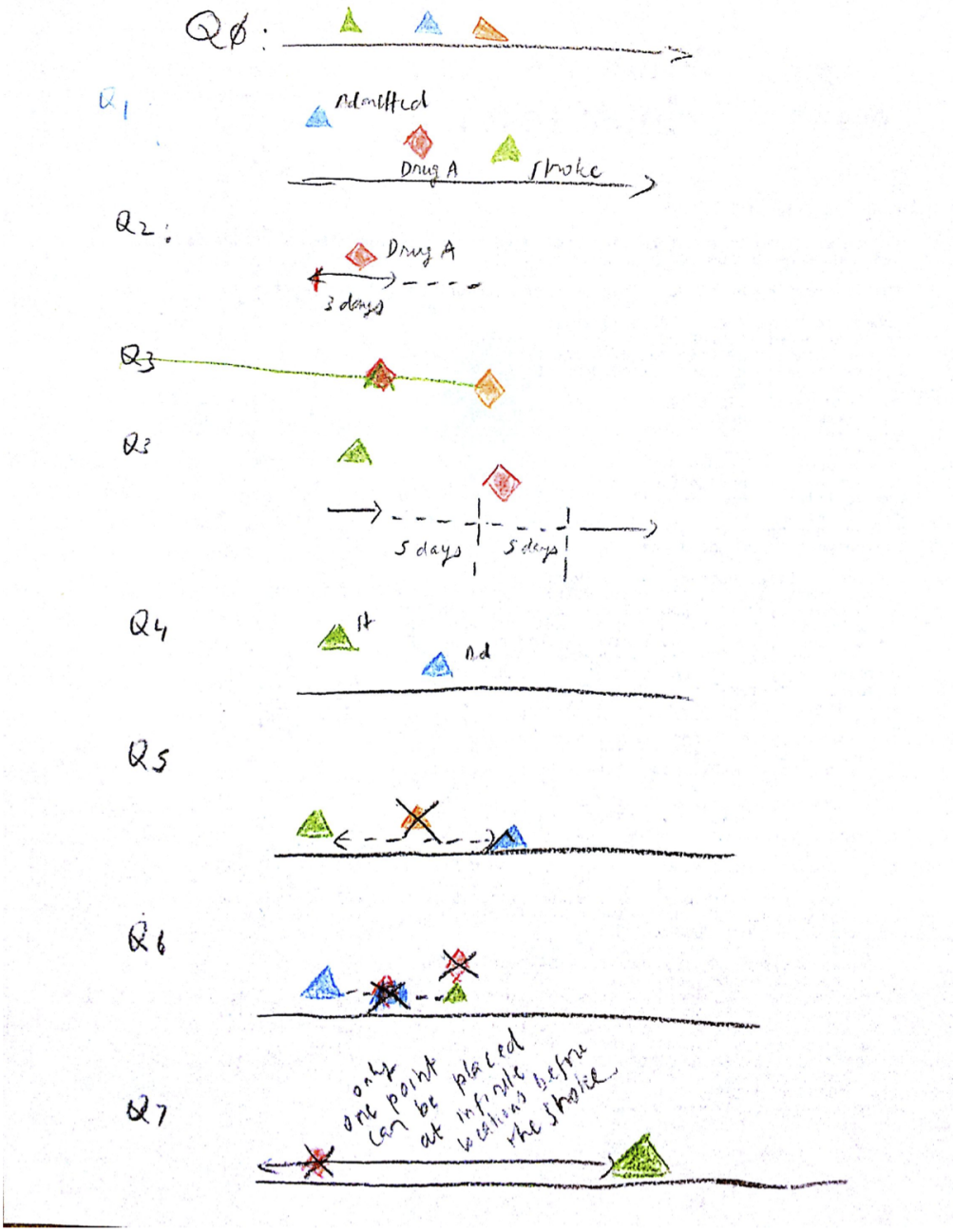
Subject 10:



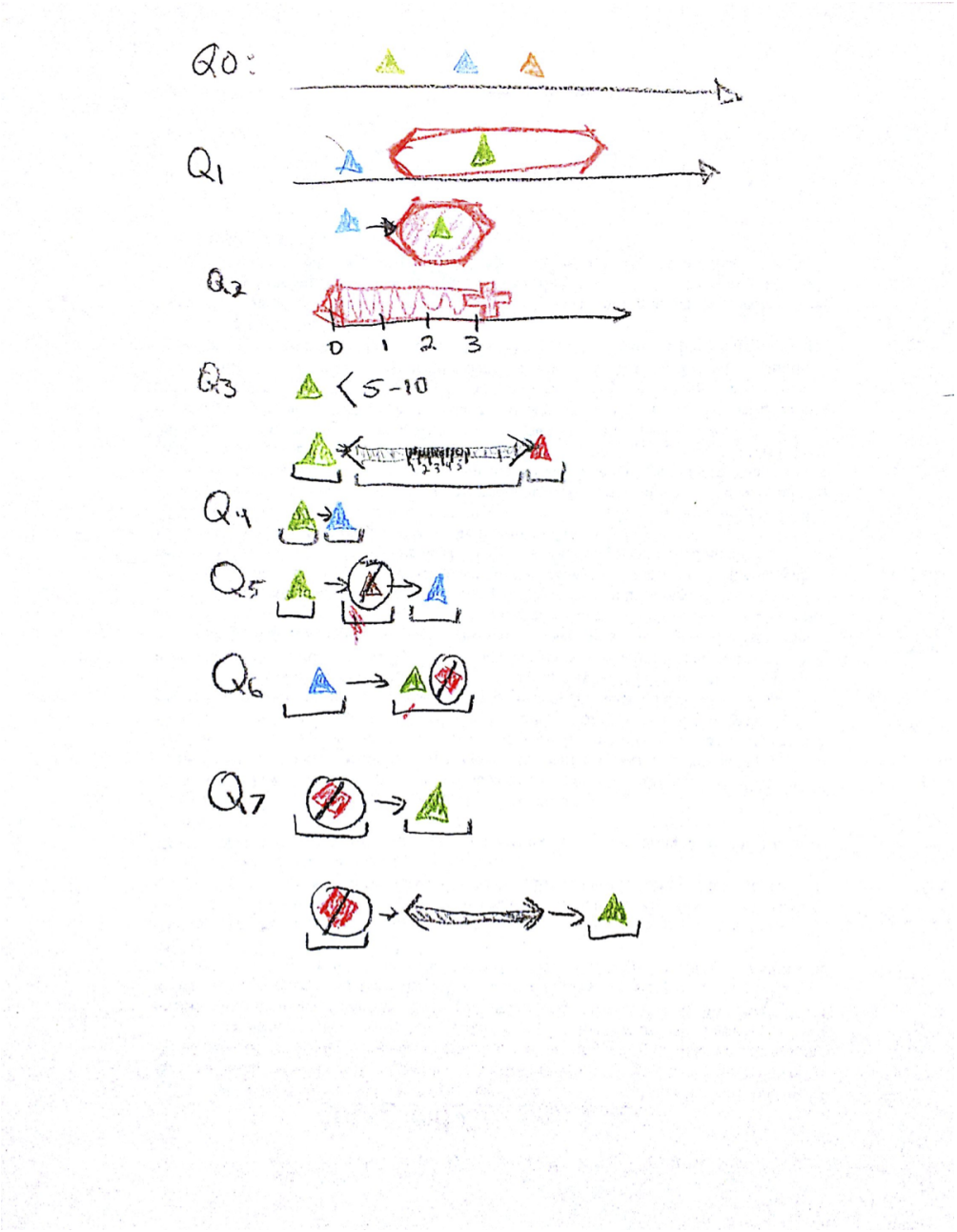
Subject 11:



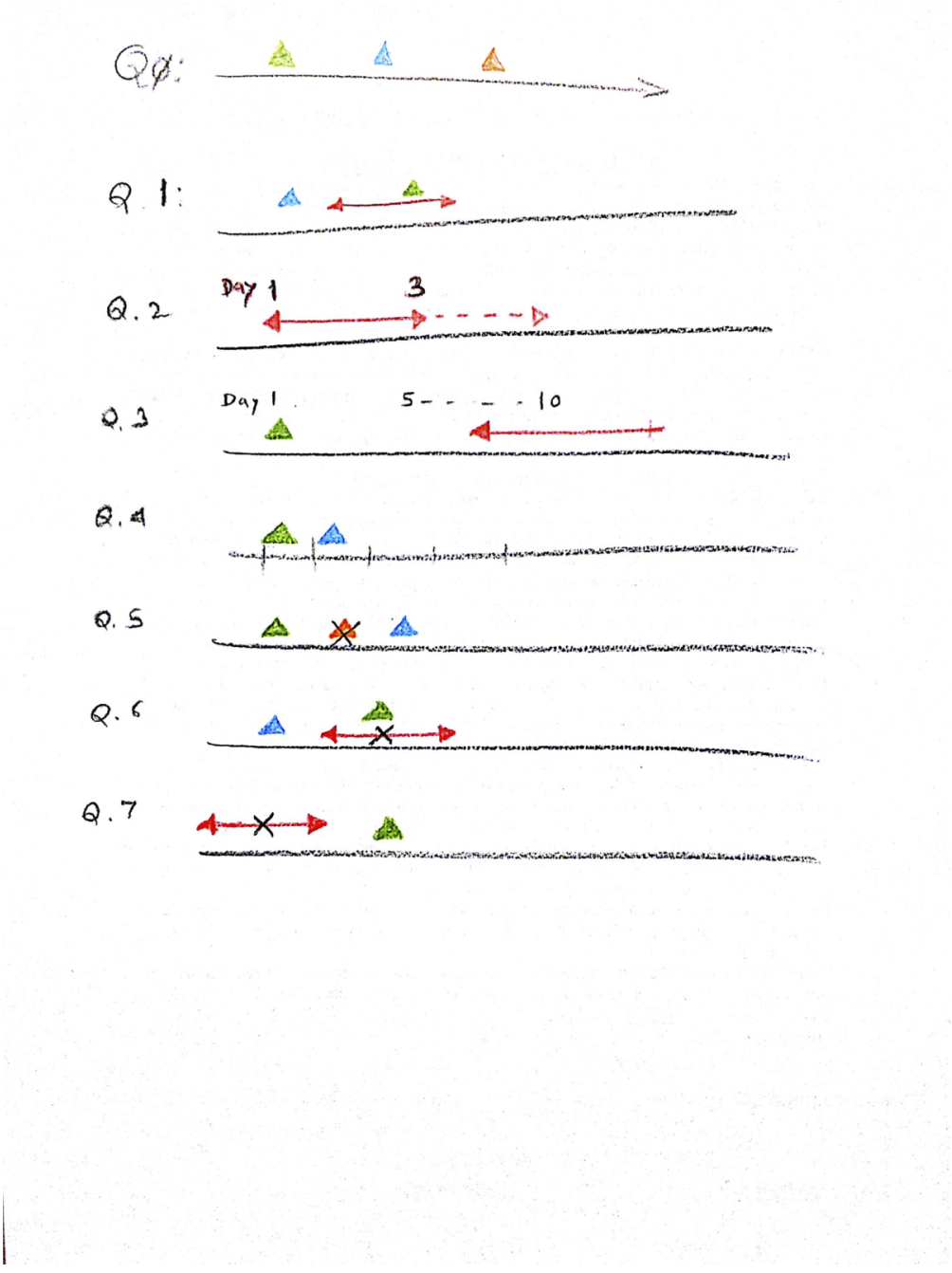
Subject 12:


Q0:
Q1
Drug A
Stroke
Q2:
Drug A
3 days
Q3
Q3
5 days
5 days
Q4
Q5
Q6
Q7
only one point can be placed at infinite distance before the stroke

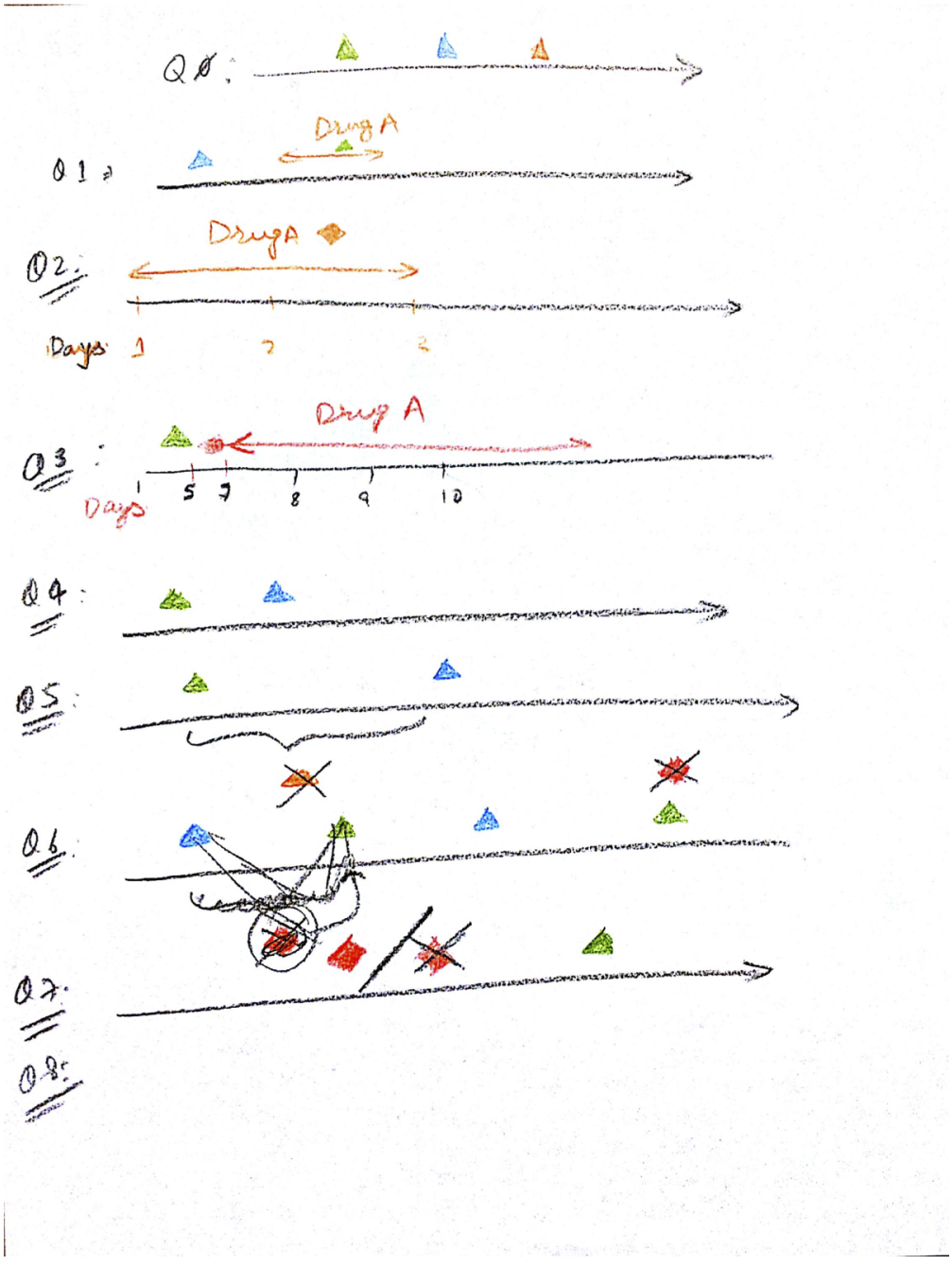
Subject 13:



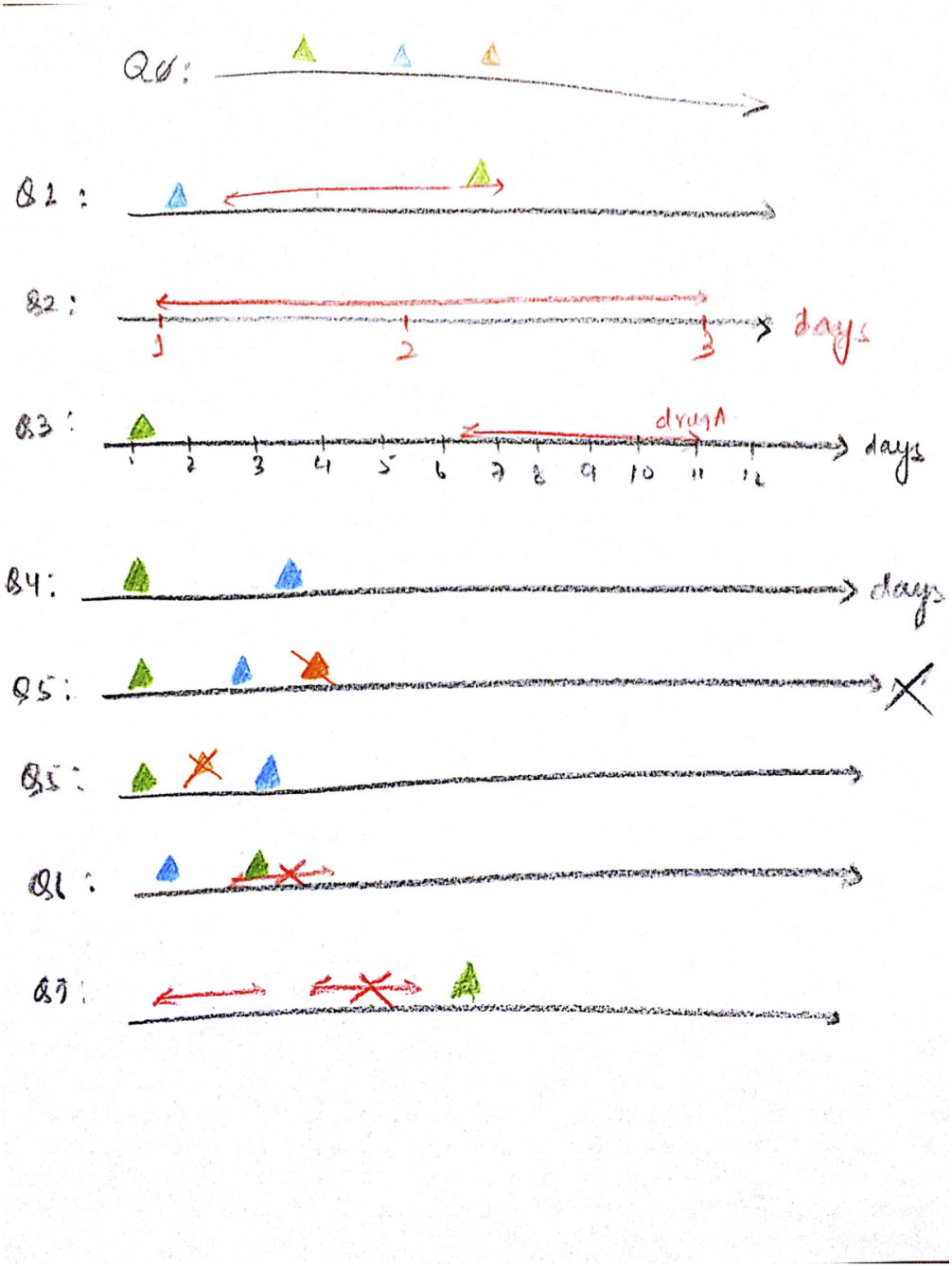
Subject 14:



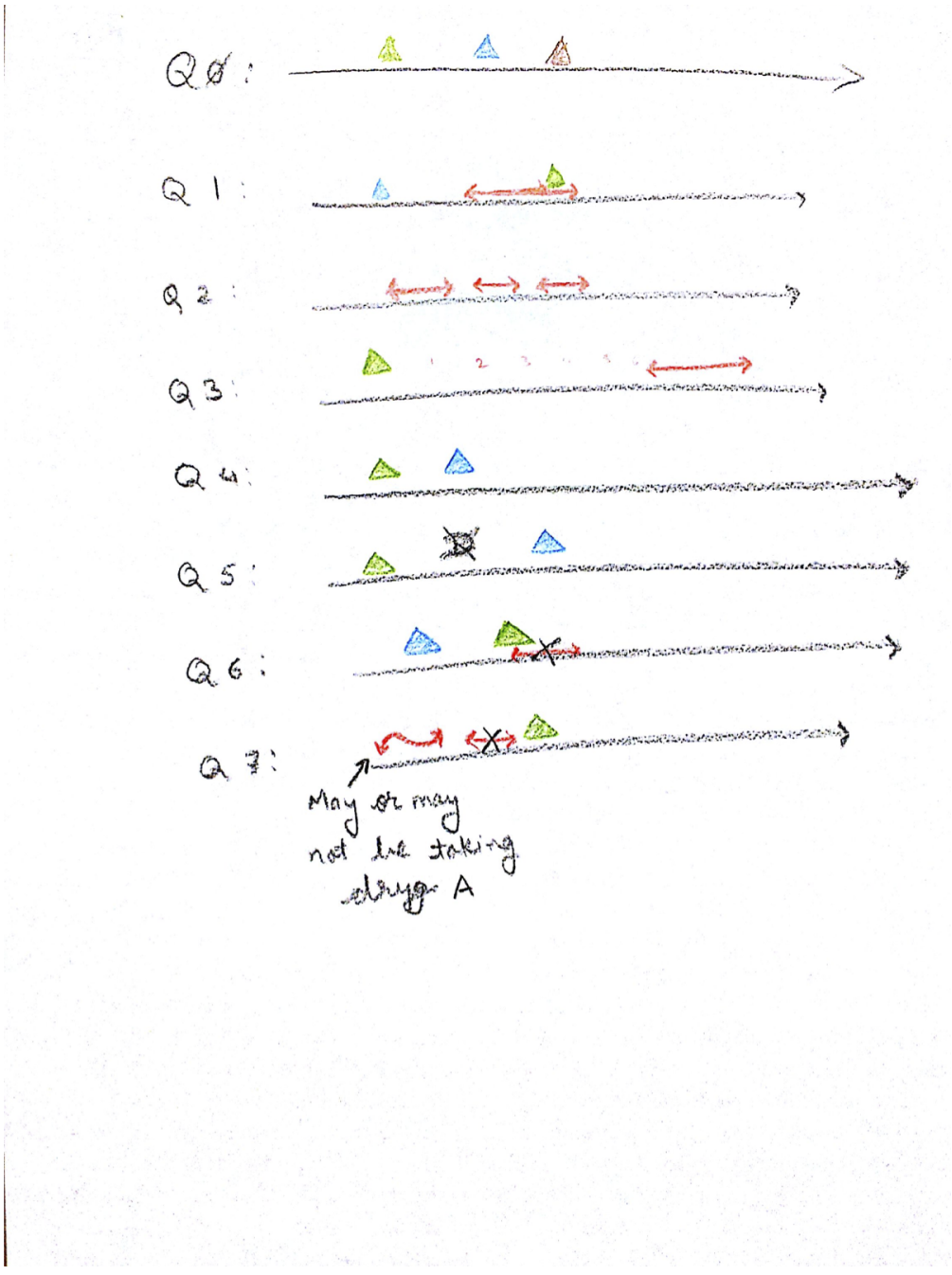
Subject 15:



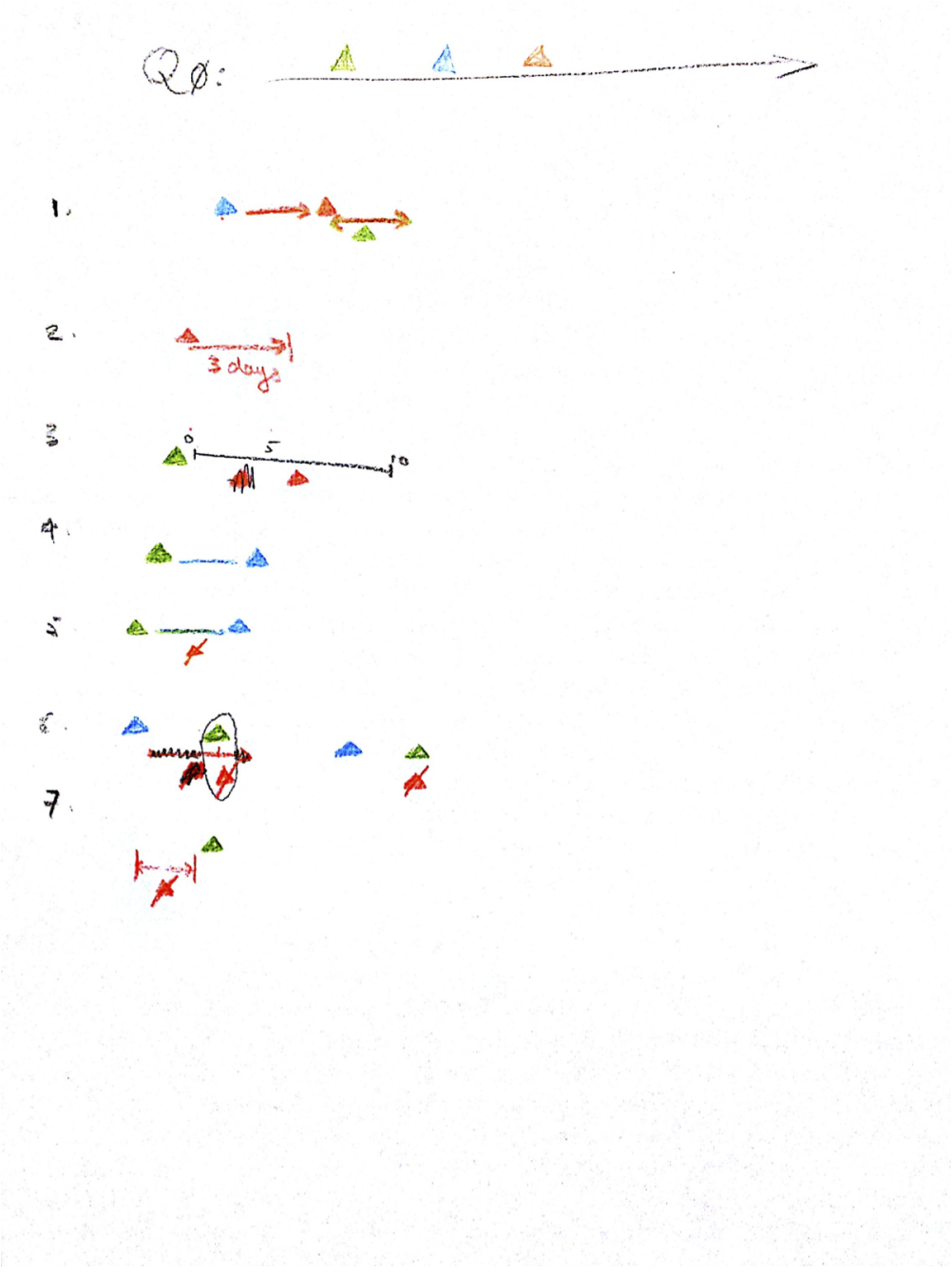
Subject 16:


Q0:
Q1:
Q2:
days
1
2
3
Q3:
drug A
days
1 2 3 4 5 6 7 8 9 10 11 12
Q4:
days
Q5:
Q6:
Q7:

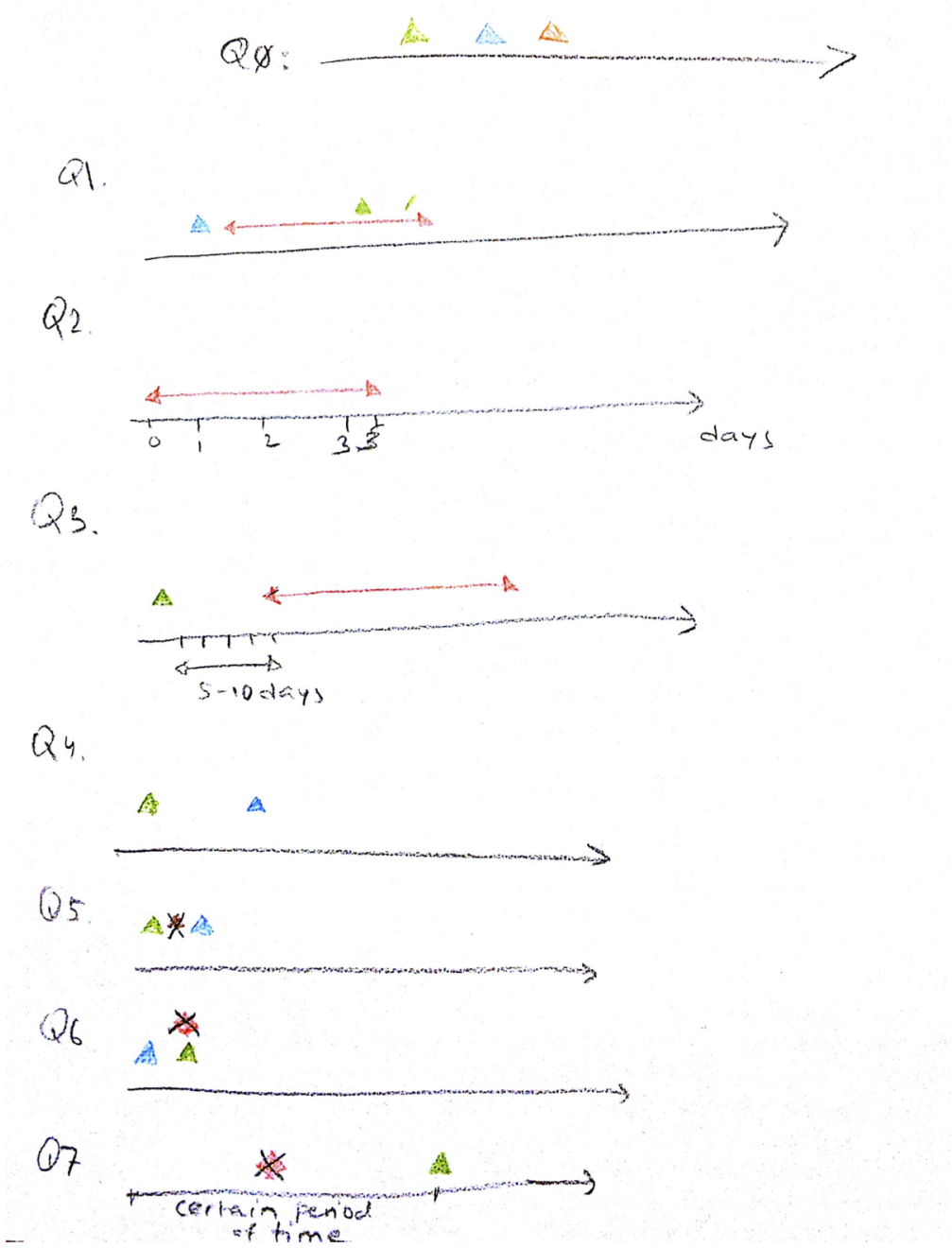
Subject 17:



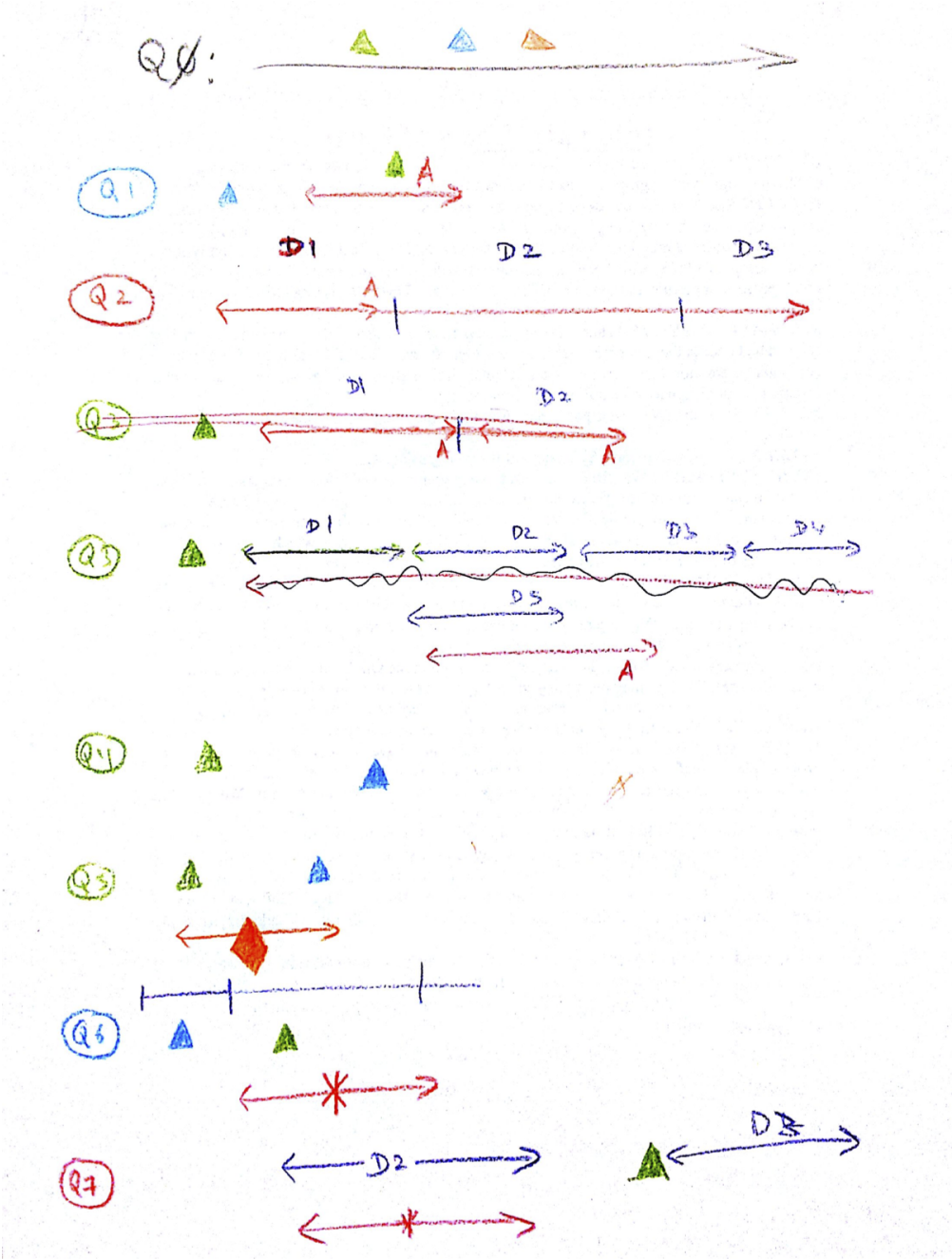
Subject 18:



Subject 19:


Q0:
Q1.
Q2.
0
1
2
3
days
Q3.
5-10 days
Q4.
Q5.
Q6
Q7
certain period of time

Subject 20:



B.4 Results

Most of the participants designed a query language that is very similar to the one used in EventFlow. Table B.1 shows all the occurrences of some significant design options and difficulties encountered by the subjects, which are described in detail below:

Absence of interval as a point (10 / 20): This is the most common characteristic. One half of the participants chose to use a negated point for representing the absence of an interval event, while the other half used negated intervals.

At least / at most visually represented (9 / 20): Almost half of the participants introduced some kind of visual element, not just text, to represent the notion of time constraints. The most common visual element was a dashed line as a way to express uncertainty.

Optional presence to specify an absence (7 / 20): Some subjects wanted to represent a gap within an interval event that is optional (the interval event may or may not occur). This complex design was presented quite often (more than expected), so perhaps the count of occurrences is biased due to the fact that Query 7 usually needed further explanation. The clarification was typically phrased as: "You are interested in all the patients that did not take the drug at SOME point. For example, the patient has been taking it for a long time, but the treatment was interrupted for just 1 day, or the patient never took Drug A before having the stroke." The concept of an interruption, as it was described in this clarification, may have led participants to include a gap in an optional interval.

Ending of an interval not specified (6 / 20): There were some participants that were very precise in the way that they wanted to define the query. More than a quarter of the subjects didn't draw the second arrow (the end-point) of the interval events if the query did not explicitly specify that the patient stopped taking Drug A.

Absence omitted (4 / 20): It was very common for participants not to

specify the absence of an event. In order to solve this issue, Query 4 and Query 5 are almost the same, but the second one includes an absence requirement. Even though this solution was very effective, there were still 4 participants who did not draw the absences until they were told to do so.

Independent absence line (2 / 20): Two participants included a new line for specifying the absence of events. Even though the idea was not very common, it was immediately understood by all of the participants when this concept was integrated into the EventFlow interface.

Interval end-point graphics (2/ 20): Some participants were very annoyed with the use of arrows for the beginning and the ending of the interval events. They interpreted this as an indication that the interval continues outwards. They suggested to use bars instead of (or with) the arrows.

Intervals as points (1 / 20): Even though there was only one case, this design decision was remarkable. The subject refused to use any representation of an interval other than the diamond icon, because he wanted to represent them as points.

A special mention has to be made for several subjects (7, 13 and 20) who presented original ideas. For example, Subject 7 introduced a consistent metaphor for representing time constraints like "at least/at most," Subject 13 represented events that happen at the same time inside a "time plate," and Subject 20 specified the time scale explicitly. After drawing and testing EventFlow, all of the participants stated that the design used in EventFlow was very easy to understand, and sometimes they found it more understandable than their own design.

Subject	Absence of interval as a point	At least / at most visually represented	Optional presence to specify an absence	Ending of an interval not specified	Absence omitted	Independent absence line	Interval end-point graphics	Intervals as points
S1	X			X				
S2		X	X					
S3		X	X	X				
S4				X	X	X		
S5	X							
S6								
S7	X	X		X		X	X	
S8		X		X				
S9		X	X	X	X			
S10	X		X		X			
S11	X	X						
S12	X	X	X					X
S13	X	X						
S14		X						
S15	X							
S16			X		X			
S17			X					
S18	X						X	
S19	X							
S20								
Total	10	9	7	6	4	2	2	1

Table B.1: Common Design options/difficulties.

Bibliography

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data, SIGMOD '08*, 2008, pp. 147 – 160.

[2] W. Aigner, S. Miksch, H. Schumann, and C. Tominski, *Visualization of Time-Oriented Data*. Springer, 2011.

[3] D. Albers, C. Dewey, and M. Gleicher, "Sequence surveyor: Leveraging overview for scalable genomic alignment visualization," in *IEEE Transactions on Visualization and Computer Graphics*. IEEE, 2011, pp. 2392 – 2401.

[4] J. Ali, R. Adam, A. K. Butler, H. Chang, M. Howard, D. Gonsalves, P. Pitt-Miller, M. Stedman, J. Winn, and J. Williams, "Trauma outcome improves following the advanced trauma life support program in a developing country," in *Journal of Trauma-Injury Infection & Critical Care*, vol. 34, no. 6, 1993, pp. 890–898.

[5] J. F. Allen, "Maintaining knowledge about temporal intervals," in *Communications of the ACM*, vol. 26, no. 11, 1983, pp. 832–843.

[6] J. F. Allen and G. Ferguson, "Actions and events in interval temporal logic," in *Journal of Logic and Computation*, vol. 4, 1994, pp. 531–579.

[7] M. Berkelaar, K. Eikland, and P. Notebaert, *lp_solve*, 5th ed., Open source (Mixed-Integer) Linear Programming system, August 2010.

[8] BestPractice, *Asthma in adults*. http://bestpractice.bmj.com/best-practice/monograph/44.html, January 2013.

[9] C. Bettini, "Time-dependent concepts: representation and reasoning using temporal description logics," in *Data & Knowledge Engineering*, vol. 22, no. 1, 1997, p. 138.

[10] J. Blaas, C. P. Botha, E. Grundy, M. W. Jones, R. S. Laramee, and F. H. Post, "Smooth graphs for visual exploration of higher-order state transitions," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6. IEEE, 2009, pp. 969 – 976.

[11] E. Carter, R. Burd, M. Monroe, C. Plaisant, and B. Shneiderman, "Using eventflow to analyze task performance during pediatric trauma resuscitation," in *AMIA Workshop on Interactive Systems in HealthCare*, 2013.

[12] CDC, *QuickStats: Number of Deaths From Poisoning, Drug Poisoning, and Drug Poisoning Involving Opioid Analgesics*. http://www.cdc.gov/mmwr/preview/mmwrhtml/mm6212a7.htm, March 2013.

[13] L. Certo, T. G. ao, and J. Borges, "Time automaton: A visual mechanism for temporal querying," in *Journal of Visual Languages & Computing*, vol. 24, no. 1, 2013, pp. 24–36.

[14] H. Chen and S. Dumais, "Bringing order to the web: automatically categorizing search results," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2000, pp. 145–152.

[15] M. Chen, M. Hearst, J. Hong, and J. Lin, "Cha-cha: A system for organizing intranet search results," in *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, 1999, pp. 11–14.

[16] Y.-L. Chena, S.-Y. Wub, and Y.-C. Wanga, "Discovering multi-label temporal patterns in sequence databases," in *Information Sciences*, vol. 181, no. 3, 2011, p. 398418.

[17] L. Chittaro and C. Combi, "Visualizing queries on databases of temporal histories: new metaphors and their evaluation," in *Data & Knowledge Engineering - Special issue: Temporal representation and reasoning*, vol. 44, no. 2, 2003, pp. 239–264.

[18] J. Chomicki, "Temporal query languages: A survey," in *Proceedings of the First International Conference on Temporal Logic, ICTL 94*, 1994, p. 506 534.

[19] C. Combi and B. Oliboni, "Visually defining and querying consistent multi-granular clinical temporal abstractions," in *Artificial intelligence in medicine*, vol. 54, no. 2, 2012, pp. 75–101.

[20] D. J. Cooka, J. C. Augustob, and V. R. Jakkula, "Ambient intelligence: Technologies, applications, and opportunities," in *Pervasive and Mobile Computing*, vol. 5, no. 4, 2009, p. 277298.

[21] S. B. Cousins and M. G. Kahn, "The visual display of temporal information," in *Artificial Intelligence in Medicine*, vol. 3, 1991, pp. 341–357.

[22] I. F. Cruz, "Doodle: a visual language for object-oriented databases," in *Proceedings of the 1992 ACM SIGMOD international conference on Management of data*. ACM, 1992, pp. 71–80.

[23] I. S. Dániel Marx, "Cleaning interval graphs," in *Algorithmica*, vol. 65, no. 2, 2013, pp. 275–316.

[24] A. K. Das and M. A. Musen, "A temporal query system for protocol-directed decision support," in *Methods of information in medicine*, vol. 33, no. 0026-1270, 1994, pp. 358–70.

[25] S. de Amo, W. P. Junior, and A. Giacometti, "Milprit*: A constraint-based algorithm for mining temporal relational patterns," in *Intercational Journal of Data Warehousing and Mining*, vol. 4, no. 4, 2008, pp. 42–61.

[26] A. M. Deshpande, C. Brandt, and P. M. Nadkarni, "Temporal query of attribute-value patient data: utilizing the constraints of clinical studies," in *International Journal of Medical Informatics*, vol. 70, 2003, pp. 59–77.

[27] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," in *Proceedings of the VLDB Endowment*. VLDB Endowment, 2008, pp. 1542–1552.

[28] D. C. Donderi, "Visual complexity: A review," in *Psychological Bulletin*, vol. 132, 2006, pp. 73–97.

[29] J. Dunn, S. Davey, A. Descour, and R. T. Snodgrass, "Sequenced subset operators: Definition and implementation," in *IEEE International Conference on Data Engineering (ICDE2002)*. IEEE, 2002, pp. 81–92.

[30] C. Dunne and B. Shneiderman, "Motif simplification: improving network visualization readability with fan, connector, and clique glyphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '13*. ACM, 2013, pp. 3247–3256.

[31] S. G. Eick and A. F. Karr, "Visual scalability," National Institute of Statistical Sciences, Research Triangle Park, NC, Tech. Rep. 106, 2002.

[32] J. A. Fails, A. Karlson, L. Shahamat, and B. Shneiderman, "A visual interface for multivariate temporal data: Finding patterns of events across multiple histories," in *Proceedings of the IEEE Symposium on Visual Analytics Science and Technology (VAST)*. IEEE, 2006, pp. 167–174.

[33] F. Ferri, P. Grifoni, and M. Rafanelli, "Querying by sketch geographical databases and ambiguities," in *Proceedings of the 16th international conference on Database and Expert Systems Applications*. Springer-Verlag, 2005, pp. 524–533.

[34] W. R. Garner, *The processing of information and structure.* New York, NY: Wiley, 1974.

[35] K. Z. Haigh and et al., "Visual query language: Finding patterns in and relationships among time series data," Lake Buena Vista, FL, 2004.

[36] A. Hakeem, Y. Sheikh, and M. Shah, "Casee: A hierarchical event representation for the analysis of videos," in *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 2004, pp. 263–268.

[37] S. Havre, B. Hetzler, and L. Nowell, "Themeriver: Visualizing theme changes over time," in *Proceedings of the IEEE Symposium on Information Visualization*. IEEE, 2000, pp. 115–123.

[38] M. A. Hearst, *Search User Interfaces*. Cambridge University Press, 2009.

[39] P. Heggernes, D. Meister, and Y. Villanger, "Induced subgraph isomorphism on interval and proper interval graphs," in *Algorithms and Computation*, vol. 6507, 2010, pp. 399–409.

[40] S. Hibino and E. A. Rundensteiner, "User interface evaluation of a direct manipulation temporal visual query language," in *MULTIMEDIA '97 Proceedings of the fifth ACM international conference on Multimedia*. ACM, 1997, pp. 99–107.

[41] H. Hochheiser, "Interactive querying of time series data," in *CHI '02 extended abstracts on Human factors in computing systems*, 2002, pp. 552–553.

[42] C. S. Jensen, J. Cliord, and S. K. G. et al., "The tsql benchmark," in *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, 1993, pp. QQ1–QQ28.

[43] J. Jin and P. Szekely, "Interactive querying of temporal data using a comic strip metaphor," in *2010 IEEE Symposium on Visual Analytics Science and Technology (VAST)*. IEEE, 2010, pp. 163–170.

[44] P.-S. Kam and A. W.-C. Fu, "Discovering temporal patterns for interval-based events," in *DaWaK 2000 Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*. Springer-Verlag, 2000, pp. 317–326.

[45] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono, "Research directions in data wrangling: visuatizations and transformations for usable and credible data," in *Information Visualization - Special issue on State of the Field and New Research Directions*, vol. 10, no. 4, 2011, pp. 271–288.

[46] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: interactive visual specification of data transformation scripts," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'11)*. ACM, 2011, pp. 3363–3372.

[47] H. A. Kautz and P. B. Ladkin, "Integrating metric and qualitative temporal reasoning," in *Proceedings of AAAI-91*, 1991, pp. 241–246.

[48] A. Kessell and B. Tversky, "Visualizing space, time, and agents: Production, performance, and preference." in *Cognitive Processing 12*, 2011, pp. 43–52.

[49] S. Kijimaa, Y. Otachib, T. Saitohc, and T. Unod, "Subgraph isomorphism in graph classes," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 312, no. 21, 2012, pp. 799–812.

[50] N. W. Kim, S. K. Card, and J. Heer, "Tracing genealogical data with timenets," in *Proceedings of the International Conference on Advanced Visual Interfaces (AVI)*. ACM, 2010, pp. 241–248.

[51] D. Knuth, J. H. Morris, and V. Pratt, "Fast pattern matching in strings," in *SIAM Journal on Computing*, vol. 6, no. 2, 1977, p. 323 350.

[52] V. Kouramajian and M. Gertz, "A visual query editor for temporal databases," in *Proceedings of OOER'95*, 1995, pp. 388–399.

[53] R. Kowalski and M. Sergot, "A logic-based calculus of events," in *New Generation Computing*, vol. 4, no. 1, 1986, pp. 67 – 95.

[54] H.-P. Kriegel, M. Pötke, and T. Seidl, "Managing intervals effciently in object-relational databases," in *VLDB 00 Proceedings of the 26th International Conference on Very Large Data Bases*, 2000, p. 407 418.

[55] P. Ladkin, "Models of axioms for time intervals," in *Proceedings of the 6th AAAI*, Seattle, WA, 1987, pp. 234–239.

[56] S. Laxman and P. S. Sastry, "A survey of temporal data mining," in *SADHANA, Academy Proceedings in Engineering Sciences*, vol. 31, 2006, p. 173 198.

[57] J. Li, A. W. chee Fu, H. He, J. Chen, H. Jin, D. McAullay, G. Williams, R. Sparks, and C. Kelman, "Mining risk patterns in medical data," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, KDD '05*, 2005, pp. 770 – 775.

[58] M. Li, M. Mani, E. A. Rundensteiner, and T. Lin, "Complex event pattern detection over streams with interval-based temporal semantics," in *Proceedings of the 5th ACM international conference on Distributed event-based system*, 2011, p. 291 302.

[59] W.-S. Li, K. Candan, K. Hirata, and Y. Hara, "Ifq: a visual query interface and query generator for object-based media retrieval," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, 1997, pp. 353–361.

[60] M. Lima, *Visual Complexity: Mapping Patterns of Information*. Princeton Architectural Press, 2011.

[61] J. Lin, M. Thomsen, and J. A. Landay, "A visual language for sketching large and complex interactive designs," in *Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves*. ACM, 2002, pp. 307–314.

[62] L. Lins, M. Heilbrun, J. Freire, and C. Silva, "Viscaretrails: Visualizing trails in the electronic health record with timed word trees, a pancreas cancer use case," in *AMIA Workshop on Visual Analytics in HealthCare*, 2013.

[63] X. Liu, M. Baumgarten, G. Smith, S. Gottlieb, C. Franey, B. Khokhar, G. Rattinger, M. Monroe, and I. Zuckerman, "Warfarin usage in elderly atrial fibrillation patients who experienced traumatic brain injury," in *66th Annual Scientific Meeting of the Gerontological Society of America*, New Orleans, LA, November 2013.

[64] L. W. MacDonald, "Using color effectively in computer graphics," in *Computer Graphics and Applications*, vol. 19, no. 4. IEEE, 1999, pp. 20–35.

[65] M. Mack and A. Olivia, "The steerable pyramid: a flexible architecture for multi-scale derivative computation," in *2nd Annual IEEE International Conference on Image Processing*, Washington, DC, 1995.

[66] ——, "Computational estimation of visual complexity," in *Poster presented at the Twelfth Annual Object, Perception, Attention, and Memory Conference*, Minneapolis, MN, 2004.

[67] H. Mannila and P. Ronkainen, "Similarity of event sequences," in *4th International Workshop on Temporal Representations and Reasoning*, 1997, pp. 136–139.

[68] B. Meindl and M. Templ, "Analysis of commercial and free and open source solvers for linear optimization problems," Vienna University of Technology, Vienna, Austria, Deliverable D3 Subtask 10, February 2012, essnet Project on Common Tools and Harmonized Methodologies for SDC in the ESS.

[69] T. Meyer, M. Monroe, C. Plaisant, R. Lan, K. Wongsuphasawat, T. Coster, S. Gold, J. Millstein, and B. Shneiderman, "Visualizing patterns of drug prescriptions with eventflow: A pilot study of asthma medications in the military health system," in *AMIA Workshop on Visual Analytics in Health-Care*, 2013.

[70] S. Michael and R. Feldman, "Visual query and exploration system for temporal relational database," in *Advances in Data Mining. Theoretical Aspects and Applications*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4597, pp. 283–295.

[71] F. Moerchen, "Algorithms for time series knowledge mining," in *KDD '06 Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 668–673.

[72] V. I. Morariu and L. S. Davis, "Multi-agent event recognition in structured scenarios," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 3289 – 3296.

[73] G. Navarro, "A guided tour to approximate string matching," in *ACM Computing Surveys (CSUR)*, 2001, pp. 31 – 88.

[74] R. Nevatia, T. Zhao, and S. Hongeng, "Hierarchical language-based representation of events in video streams," in *Proceedings of the IEEE Workshop on Event Mining*. Los Alamitos, CA: IEEE, 2003.

[75] S. J. Nordbø, "Information visualisation and the electronic health record," in *Masters Thesis: Norwegian University of Science and Technology*, 2006.

[76] H. Obweger, M. Suntinger, J. Schiefer, and G. Raidl, "Similarity searching in sequences of complex events," in *Proc. International Conf. on Research Challenges in Information Science (RCIS)*. IEEE, 2010, pp. 631–640.

[77] E. Onukwugha, Y. Kwok, C. Young, C. Mullins, B. Seal, and A. Hussain, "Variation in the length of radiation therapy among men diagnosed with incident metastatic prostate cancer," in *55th Annual Meeting of the American Society for Radiation Oncology*, Atlanta, GA, September 2013.

[78] D. Patel, W. Hsu, and M. L. Lee, "Mining relationships among interval-based events for classification," in *SIGMOD '08 Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 393–404.

[79] D. Patnaik, P. Butler, N. Ramakrishnan, L. Parida, B. J. Keller, and D. A. Hanauer, "Experiences with mining temporal event sequences from electronic medical records: initial successes and some challenges," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 360–368.

[80] A. Perer and B. Shneiderman, "Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08*, 2008, pp. 265–274.

[81] ——, "The importance of integrating statistics and visualization: Long-term case studies supporting exploratory data analysis of social networks," in *IEEE Computer Graphics & Applications*, vol. 29, no. 3, 2009, pp. 39–51.

[82] C. Plaisant, S. Lam, B. Shneiderman, M. S. Smith, D. Roseman, G. Marchand, M. Gillam, C. Feied, J. Handler, and H. Rappaport, "Searching electronic health records for temporal patterns in patient histories: A case study with microsoft amalga," in *Proceedings of the AMIA Annual Symposium*, 2008, pp. 601–605.

[83] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Shneiderman, "Lifelines: Visualizing personal histories," in *ACM CHI '96 Conference Proc.* ACM, 1996, pp. 221–227.

[84] A. R. Post, T. Kurc, R. Willard, H. Rathod, M. Mansour, A. K. Pai, W. M. Torian, S. Agravat, S. Sturm, and J. H. Saltz, "Temporal abstraction-based clinical phenotyping with eureka!" in *Proceedings of AMIA Annual Symposium*, 2013.

[85] W. Pratt, "Dynamic organization of search results using the umls," in *Journal of the American Medical Informatics Association*, 1997, pp. 480–484.

[86] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction.* Springer-Verlag, 1985.

[87] J. Renz, "Implicit constraints for qualitative spatial and temporal reasoning," in *Proceedings of Knowledge Representation*, 2012.

[88] A. Rind, T. Wang, W. Aigner, S. Miksch, K. Wongsuphasawat, C. Plaisant, and B. Shneiderman, "Interactive information visualization for exploring and querying electronic health records: A systematic review," University of Maryland, College Park: Human Computer Interaction Lab (HCIL), Tech. Rep., 2010.

[89] R. Rosenholtz, Y. Li, J. Mansfield, and Z. Jin, "Feature congestion: a measure of display clutter," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '05*, 2005, pp. 761–770.

[90] R. Rosenholtz, Y. Li, and L. Nakano, "Measuring visual clutter," in *Journal of Vision*, vol. 7, no. 17, 2007, pp. 1 – 22.

[91] K. Ryall, N. Lesh, T. Lanning, D. Leigh, H. Miyashita, and S. Makino, "Querylines: approximate query for visual browsing," in *CHI '05 extended abstracts on Human factors in computing systems.* New York, NY, USA: ACM, 2005, pp. 1765–1768.

[92] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi, "Expressing and optimizing sequence queries in database systems," in *ACM Transactions on Database Systems (TODS)*, vol. 29, no. 2, 2004, p. 282 318.

[93] M. Saeed and R. Mark, "A novel method for the efficient retrieval of similar multiparameter physiologic time series using wavelet-based symbolic representations." in *Proceedings of the AMIA Annual Symposium*, 2006, p. 679683.

[94] H. Samet, "Pictorial query specification for browsing through spatially referenced image databases," in *Journal of Visual Languages and Computing*, 1998, pp. 567–596.

[95] M. Sedlmair, M. D. Meyer, and T. Munzner, "Design study methodology: Reflections from the trenches and the stacks," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, 2012, pp. 2431–2440.

[96] J. Seo and B. Shneiderman, "Knowledge discovery in high-dimensional data: case studies and a user survey for the rank-by-feature framework," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 3, 2006, pp. 311 – 322.

[97] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *VL '96 Proceedings of the 1996 IEEE Symposium on Visual Languages*, 1996, p. 336.

[98] B. Shneiderman and C. Plaisant, "Strategies for evaluating information visualization tools," in *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization (BELIV '06)*. ACM, 2006, pp. 1–7.

[99] J. G. Snodgrass and M. Vanderwart, "A standardized set of 260 pictures: Norms for name agreement, image agreement, familiarity, and visual complexity," in *Journal of Experimental Psychology:Human Learning and Memory*, vol. 6, no. 2, 1980, pp. 174–215.

[100] R. Snodgrass, "The temporal query language tquel," in *ACM Transactions on Database Systems (TODS)*, vol. 12, no. 2. ACM, 1987, pp. 247–298.

[101] R. T. Snodgrass, *The TSQL2 Temporal Query Language*. Springer, 1995.

[102] ——, *Developing Time-Oriented Database Applications in SQL*. San Francisco, CA: Morgan Kaufmann, 1999.

[103] F. Song and R. Cohen, "The interpretation of temporal relations in narrative," in *Proceedings of the 7th AAAI*, St. Paul, MN, 1988, pp. 745–750.

[104] *Advanced Trauma Life Support Course for Physicians*, Subcommittee on Advanced Trauma Life Support, Chicago, IL, 1992.

[105] J. Sun, F. Wang, J. Hu, and S. Edabollahi, "Visual cluster analysis in support of clinical decision intelligence," in *AMIA Annual Symposium Proceedings*, 2011, p. 481490.

[106] ——, "Supervised patient similarity measure of heterogeneous patient records," in *SIGKDD ACM Special Interest Group on Knowledge Discovery in Data*, vol. 14, no. 1. ACM, 2012, pp. 16–24.

[107] D. Toman, "Point vs. interval-based query languages for temporal databases," in *Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, 1996, pp. 58 – 67.

[108] H. Y. Tsang, M. Tory, and C. Swindells, "eseetrack visualizing sequential fixation patterns," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6. IEEE, 2010, pp. 953–962.

[109] E. Tufte, *The Visual Display of Quantitative Information*. Graphic Press, 1983.

[110] G. D. van Olden, J. D. Meeuwis, H. W. Bolhuis, and H. B. abd R. Jan A. Goris, "Clinical impact of advanced trauma life support," in *American Journal of Emergency Medicine*, vol. 22, no. 7, 2004, pp. 522–525.

[111] M. Vilain, H. Kautz, and P. van Beek, "Constraint propagation algorithms for temporal reasoning: a revised report," in *Readings in qualitative reasoning about physical systems*. Morgan Kaufmann Publishers Inc., 1990, pp. 373–381.

[112] P. C. Vitz, "Preference for different amounts of visual complexity," in *Behavioral Science*, vol. 11, no. 2, 1966, p. 105114.

[113] K. Vrotsou, "Everyday mining: Exploring sequences in event-based data," in *Ph.D. Dissertation from the Department of Science and Technology, Linköping University*, 2010.

[114] K. Vrotsou and C. Forsell, "A qualitative study of similarity measures in event-based data," in *Proc. Human Interface and the Management of Information. Interacting with Information Symp. on Human Interface.* Springer, 2011, pp. 170–179.

[115] K. Vrotsou, J. Johansson, and M. Cooper, "Activitree: Interactive visual exploration of sequences in event-based data using graph similarity," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, 2000, pp. 945–952.

[116] T. D. Wang, "Interactive visualization techniques for searching temporal categorical data," in *Ph.D. Dissertation from the Department of Computer Science, University of Maryland*, 2010.

[117] T. D. Wang, A. Deshpande, and B. Shneiderman, "A temporal pattern search algorithm for personal history event visualization," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, 2012, pp. 799 – 812.

[118] T. D. Wang, C. Plaisant, A. J. Quinn, R. Stanchak, B. Shneiderman, and S. Murphy, "Aligning temporal data by sentinel events: Discovering patterns in electronic health records," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 457–466.

[119] T. D. Wang, K. Wongsuphasawat, C. Plaisant, and B. Shneiderman, "Extracting insights from electronic health records: case studies, a visual analytics process model, and design recommendations," in *Journal of Medical Systems*, vol. 35, no. 5, 2011, pp. 1135–1152.

[120] C. Ware, *Information Visualization: Perception for Design.* Morgan Kaufmann, 2004.

[121] M. Wattenberg, "Sketching a graph to query a time-series database," in *CHI '01 extended abstracts on Human factors in computing systems*. ACM, 2001, pp. 381–382.

[122] K. Wongsuphasawat, "Interactive exploration of temporal event sequences," in *Ph.D. Dissertation from the Department of Computer Science, University of Maryland*, 2012.

[123] K. Wongsuphasawat, J. A. G. Gómez, C. Plaisant, T. D. Wang, M. Taieb-Maimon, and B. Shneiderman, "Lifeflow: Visualizing an overview of event sequences," in *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems (CHI'11)*. ACM, 2011, pp. 1747–1756.

[124] K. Wongsuphasawat, C. Plaisant, M. Taieb-Maimon, and B. Shneiderman, "Querying event sequences by exact match or similarity search: Design and empirical evaluation," in *Interacting with Computers*, vol. 24, no. 2, 2012, pp. 55–68.

[125] H. Wu, B. Salzberg, G. C. Sharp, S. B. Jiang, H. Shirato, and D. Kaeli, "Subsequence matching on structured time series data," in *SIGMOD '05 Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 682–693.

[126] S.-Y. Wu and Y.-L. Chen, "Mining nonambiguous temporal patterns for interval-based events," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 6. IEEE, 2007, pp. 742–758.

[127] S.-Y. Wua and Y.-L. Chenb, "Discovering hybrid temporal patterns from sequences consisting of point- and interval-based events," in *Data & Knowledge Engineering*, vol. 68, no. 11, 2009, p. 13091330.

[128] C. Zaniolo, "Event-oriented date models and temporal queries in transaction-time databases," in *16th International Symposium on Temporal Representation and Reasoning*, 2009, pp. 47 – 53.